Perceptron Training Algorithm

Introduction

- Implemented perceptron training algorithm and applied it on the following problems :
  - 2-input NAND, NOR
  - 2-input XOR
  - 5-input palindrome
  - 5-input majority
  - 6-input parity
- Implemented cycle detection in cases where PTA does not converge.

Implementation

- Given the function, generate the truth table of the function and the corresponding vectors.

- Separate out the vectors in two classes viz. zero-class and one-class and add appropriate labels to the vectors and negate the vectors of zero-class.

- Function *check(w)* : where w is a weight vector.
  - Checks whether $w^T x_i > 0$, for every x_i in vector set.
  - That is, whether convergence has been reached.
  - Returns x_i for which the check fails.

Algorithm

```
w=zero-vector // weight vector initialization
begin
        if max # iterations is reached
           print 'No convergence'
           return
        temp = check(w)
        if temp==[ ]
           print "convergence reached"
           return w
        else
           if cycle is detected
              find length of the cycle
              return
           else
              w = w + temp
              repeat
end
```

Cycle Detection

- Cycle is detected using slow-fast method.
- Two weight vectors are kept, '*wslow*' and '*wfast*'.
- On failing, *wfast* is modified twice whereas *wslow* is modified only once.

```
// slow

temp = check(wslow)
wslow = wslow + temp
```

```
// fast
temp = check(wfast)
wfast = wfast + temp
temp = check(wfast)
wfast = wfast + temp
```

- If at some point, wslow and wfast become same, then the cycle is detected.

Finding Cycle Length

- After the cycle is detected, store the weight vector(w), say in *stored_weight*.
- Now keep iterating, modifying w every time until it becomes equal to *stored_weight* again.
- Count the number of steps it took for the above to happen. This gives the cycle length.

Observations : Convergence Examples

- 2-NAND
  - # steps in convergence = 19
  - Weight vector = [-4, -3, -2]
- 2-NOR
  - # steps in convergence = 10
  - Weight vector = [-1, -2, -2]
- 5-Majority
  - # steps in convergence = 68
  - Weight vector = [9, 4, 4, 3, 3]

Observations : Non Convergence Examples

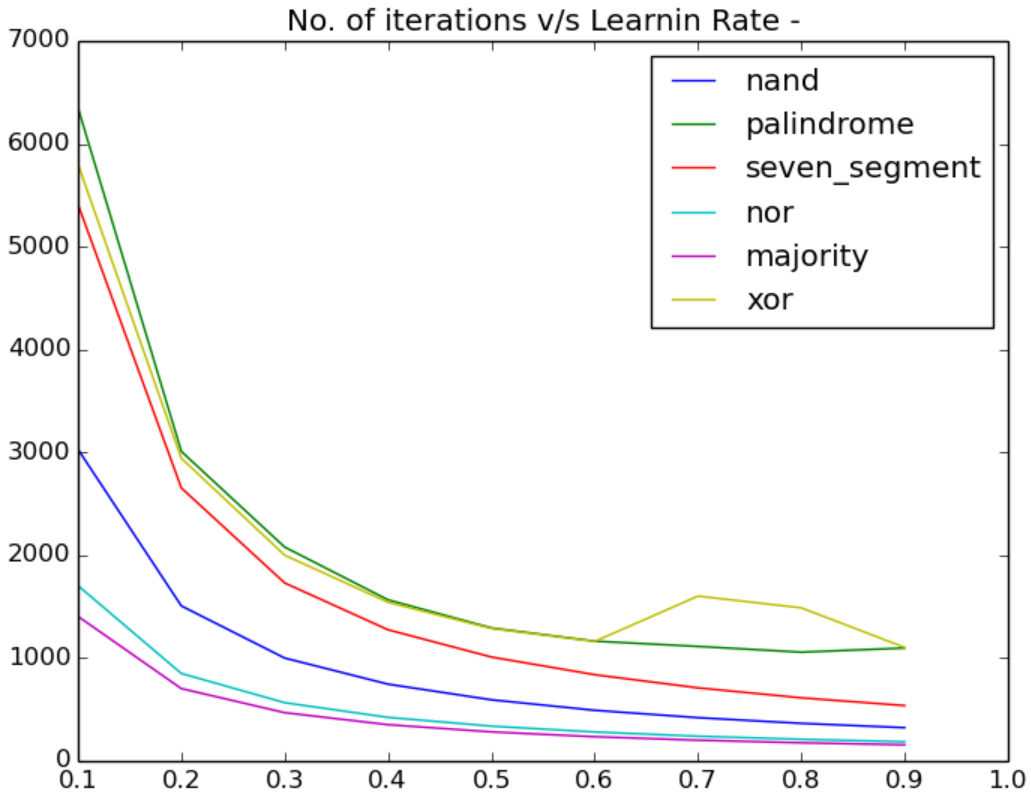
- 2-XOR
  - Cycle detected of length 4
- 5-Palindrome
  - Cycle detected of length 4
- 6-Parity
  - Cycle detected of length 4

Feed Forward And Back Propagation

Introduction

- Implemented feed forward neural network with back propagation for the following problems :
  - 2-input NAND
  - 2-input XOR
  - 5-input palindrome
  - 5-input majority
  - 5-input parity
  - Seven segment display
  - Twitter sentiment Analysis
  - IRIS and MONK data sets

Plot for Number of Iterations with Learning Rate

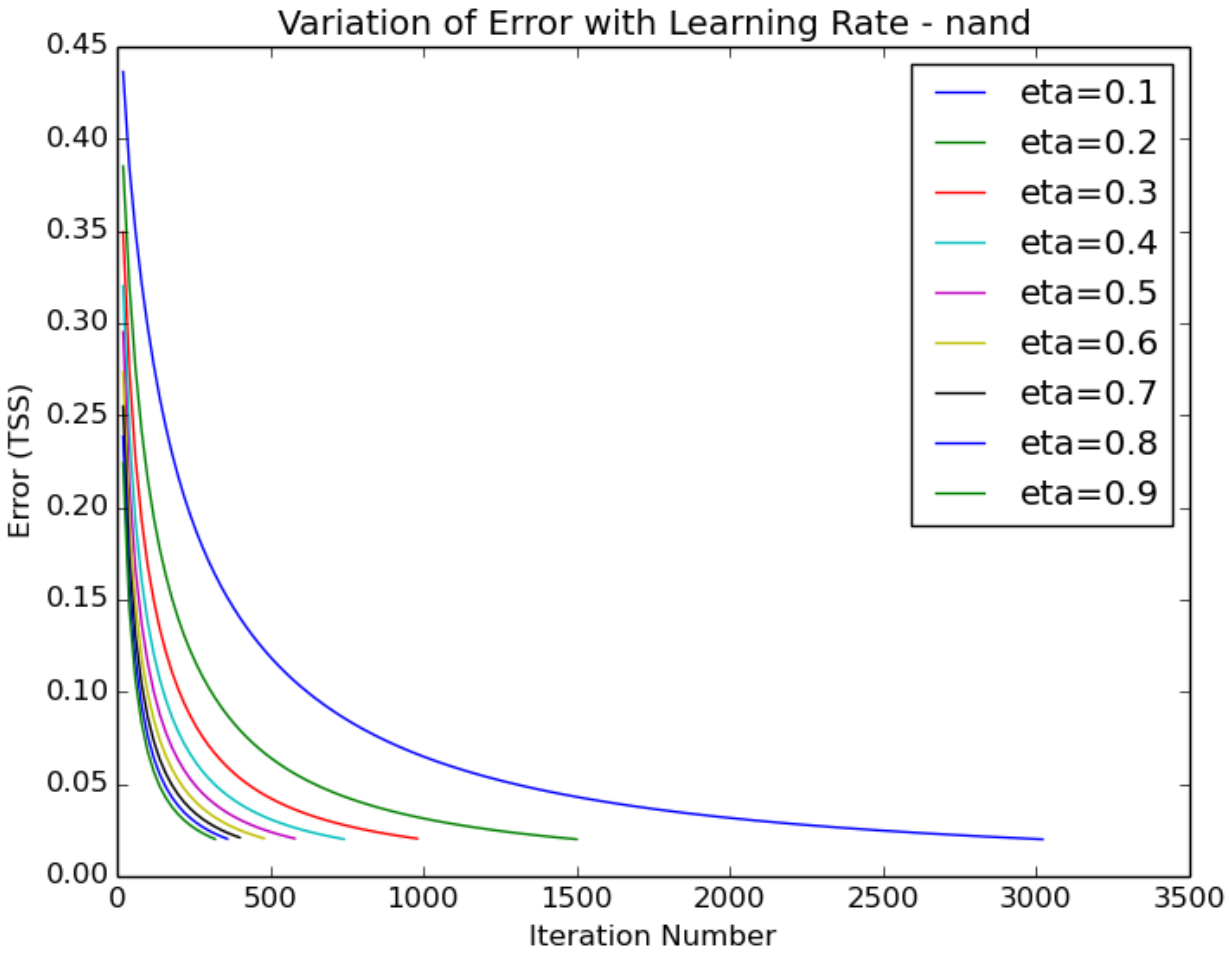


No. of iterations v/s Learning rate

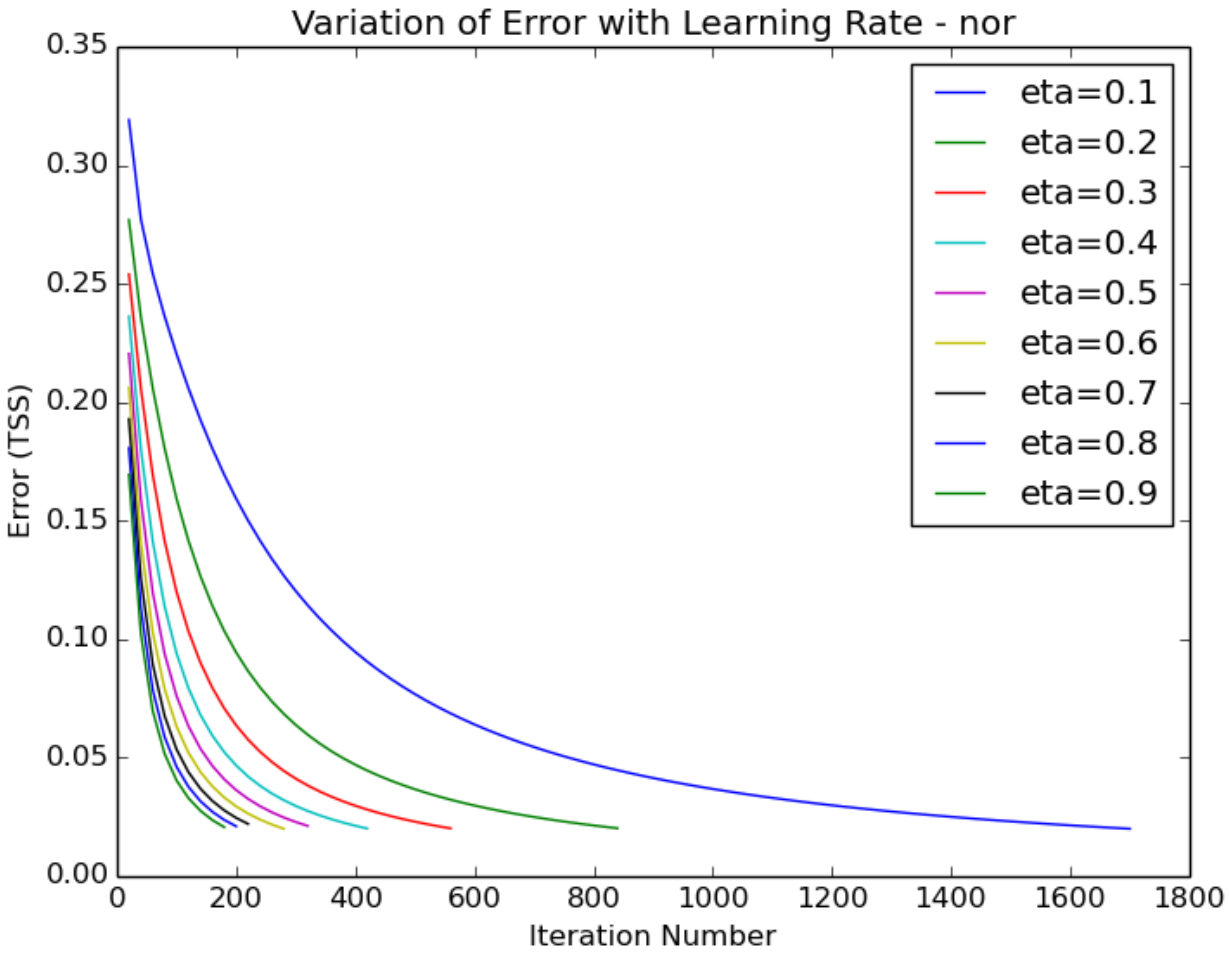
It is observed from the graph that the number of iterations for the convergence of FFNN decreases as the learning rate increases.

Plots for Variation of Error with Learning Rate

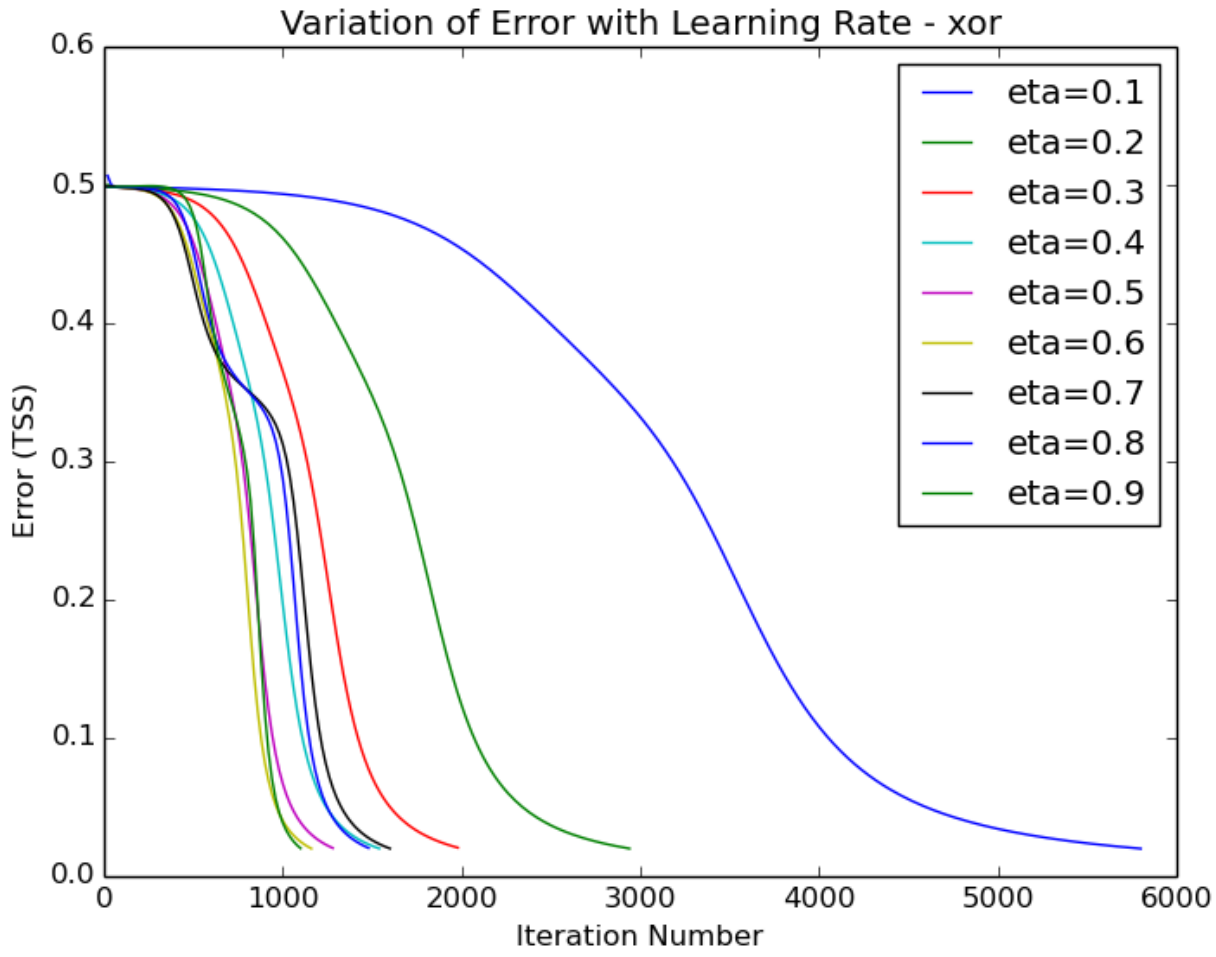
2-input NAND


Variation of Error with Learning Rate - nand
Error (TSS)
Iteration Number
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
500
1000
1500
2000
2500
3000
3500
eta=0.1
eta=0.2
eta=0.3
eta=0.4
eta=0.5
eta=0.6
eta=0.7
eta=0.8
eta=0.9

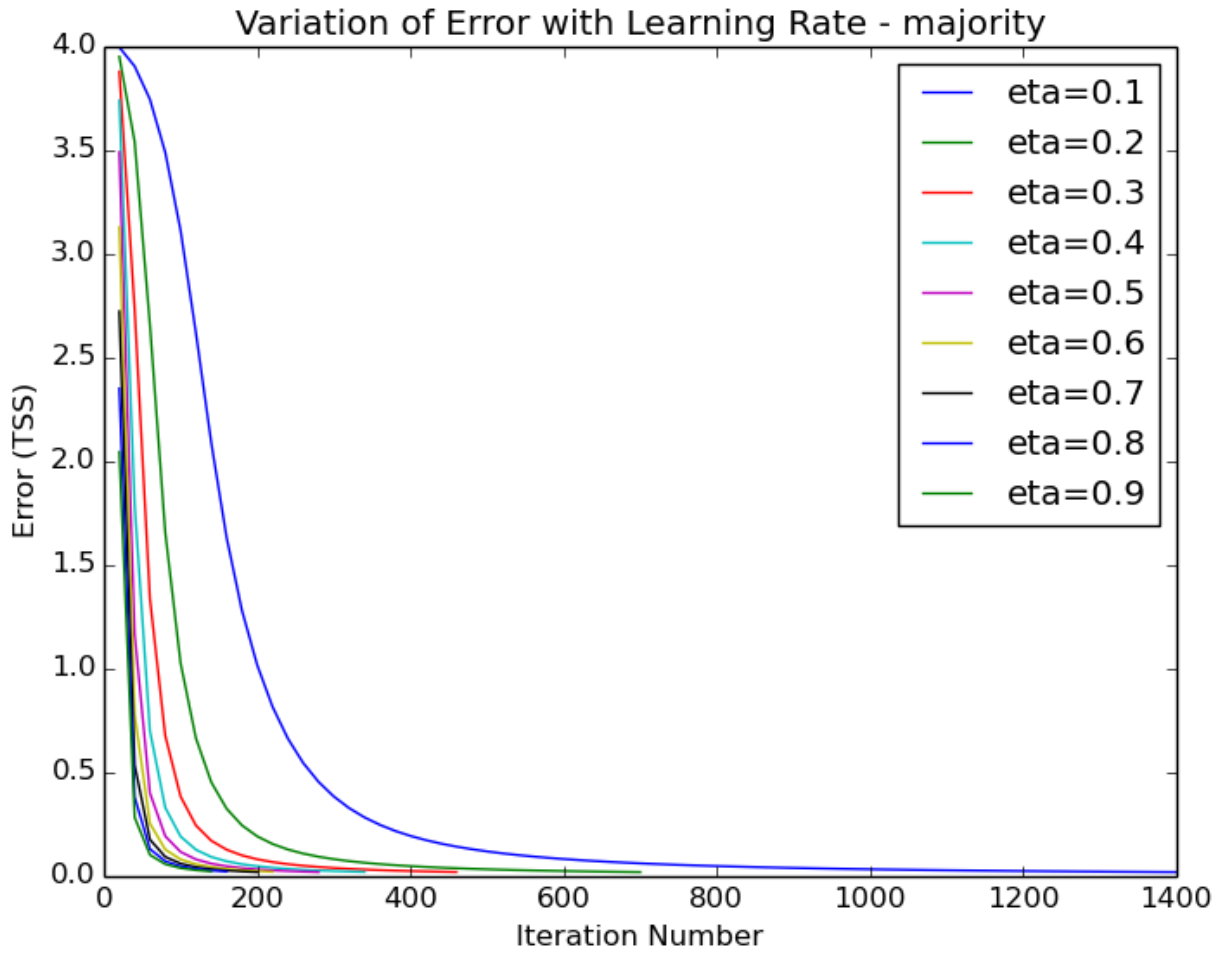
2-Input NOR


Variation of Error with Learning Rate - nor
Error (TSS)
Iteration Number
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0
200
400
600
800
1000
1200
1400
1600
1800
eta=0.1
eta=0.2
eta=0.3
eta=0.4
eta=0.5
eta=0.6
eta=0.7
eta=0.8
eta=0.9

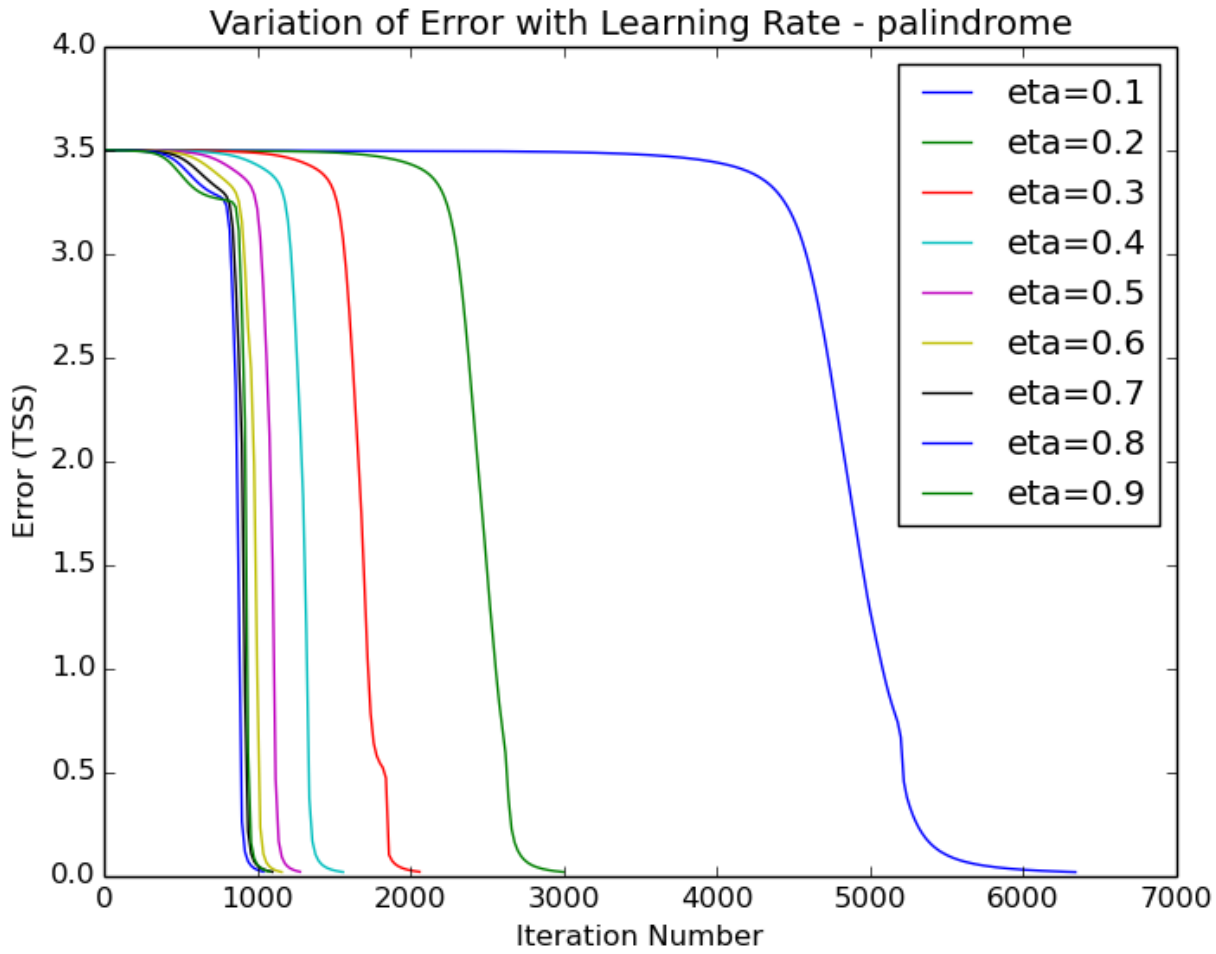
2-Input XOR


Variation of Error with Learning Rate - xor
Error (TSS)
Iteration Number
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
1000
2000
3000
4000
5000
6000
eta=0.1
eta=0.2
eta=0.3
eta=0.4
eta=0.5
eta=0.6
eta=0.7
eta=0.8
eta=0.9

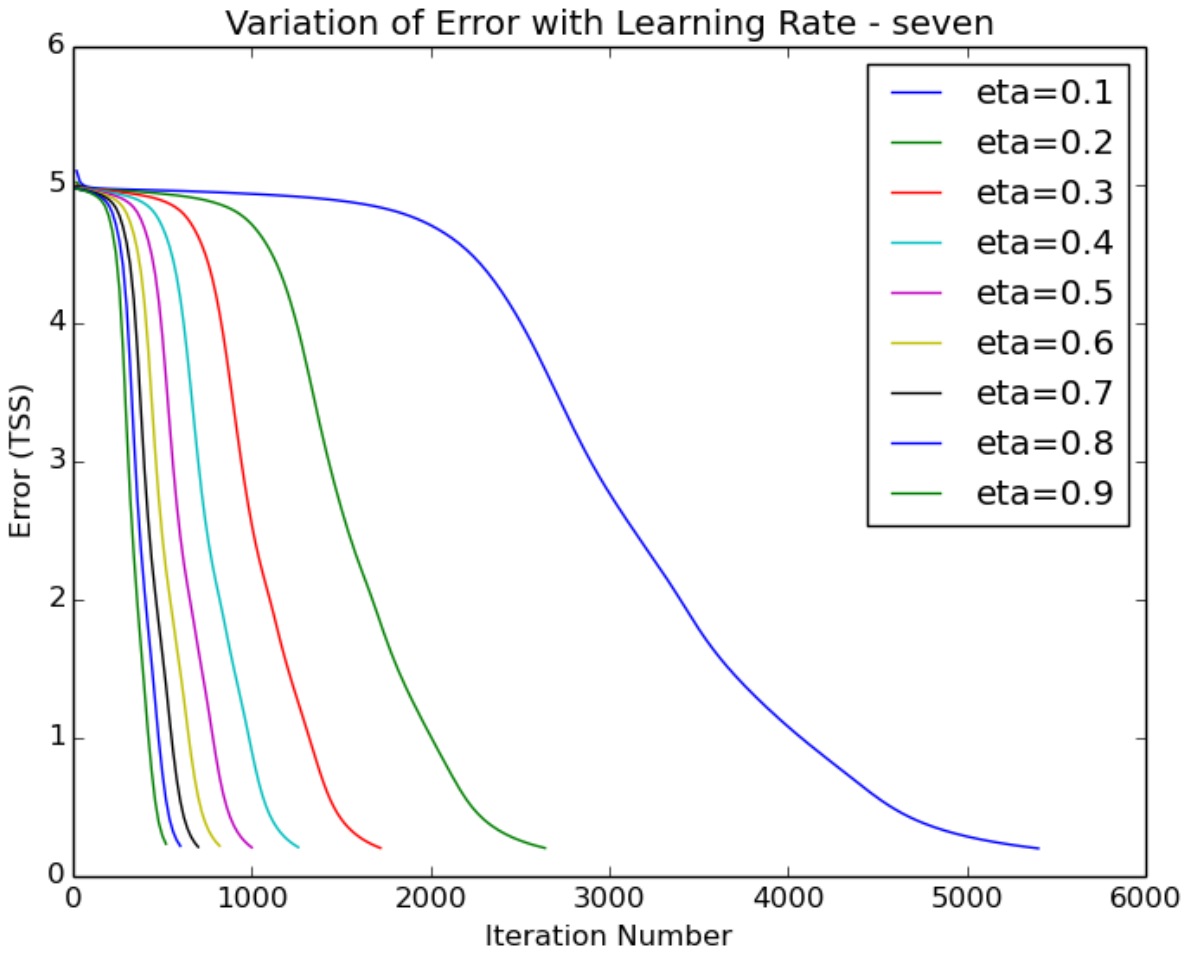
5-input Majority



5-Input Palindrome



Seven Segment Display



Local Minima and Saturation

- The back propagation algorithm can get stuck in the local minimum. This is because back propagation algorithm tries greedily to reduce the error by updating weights.

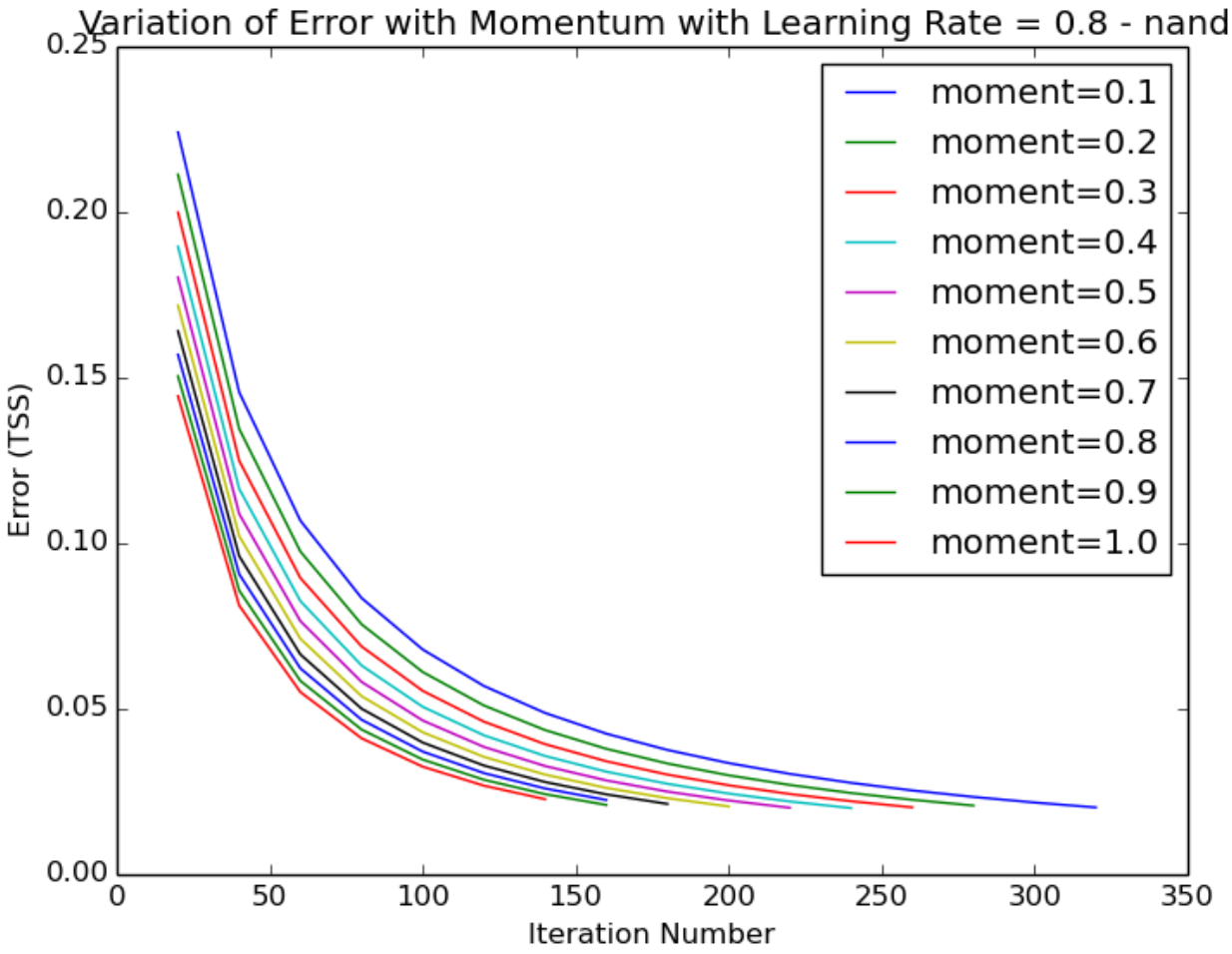
- In such case, the error does not reduce below the local minimum value , hence BP goes into a state of saturation, thereby not attaining global minimum.

Momentum

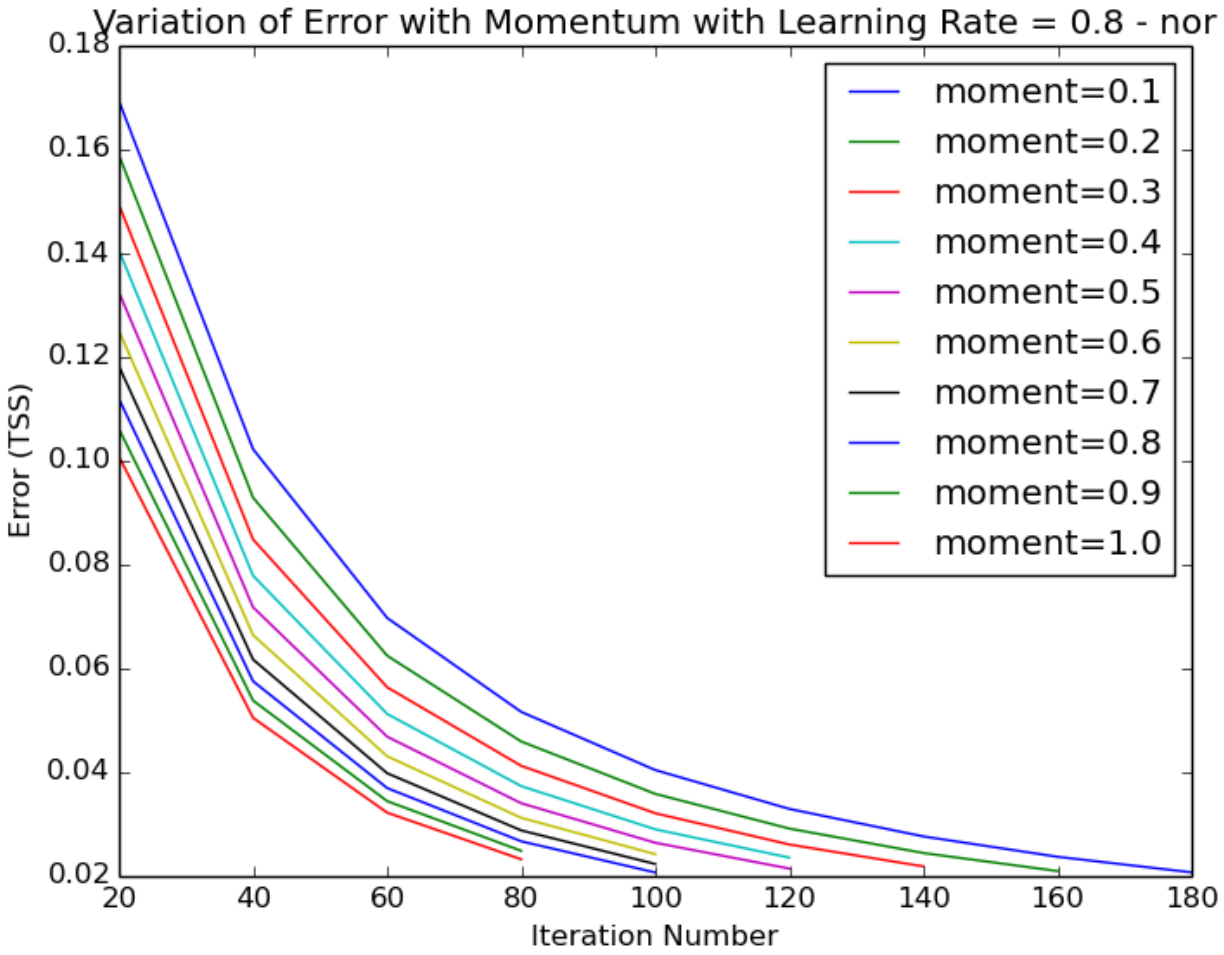
- To avoid saturation, momentum factor is introduced in the algorithm which while updating weights also takes into account the weight updates in the previous iteration.
- It helps to avoid the local minimum by accelerating the movement out of the trough and dampening the oscillations inside the trough.

Plots for Variation of Error with Moment

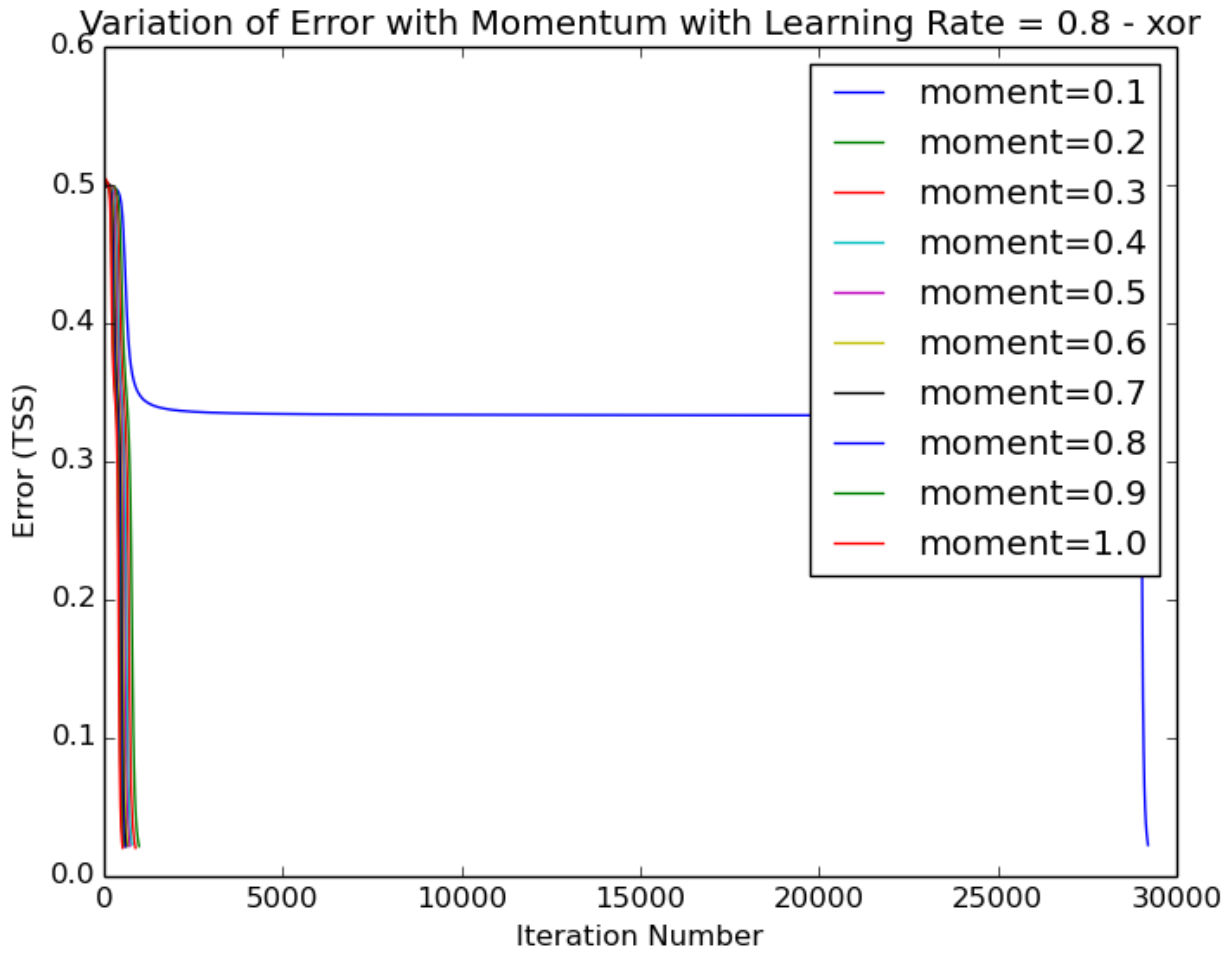
2-Input NAND


Variation of Error with Momentum with Learning Rate = 0.8 - nand
Error (TSS)
Iteration Number
0.25
0.20
0.15
0.10
0.05
0.00
0
50
100
150
200
250
300
350
moment=0.1
moment=0.2
moment=0.3
moment=0.4
moment=0.5
moment=0.6
moment=0.7
moment=0.8
moment=0.9
moment=1.0

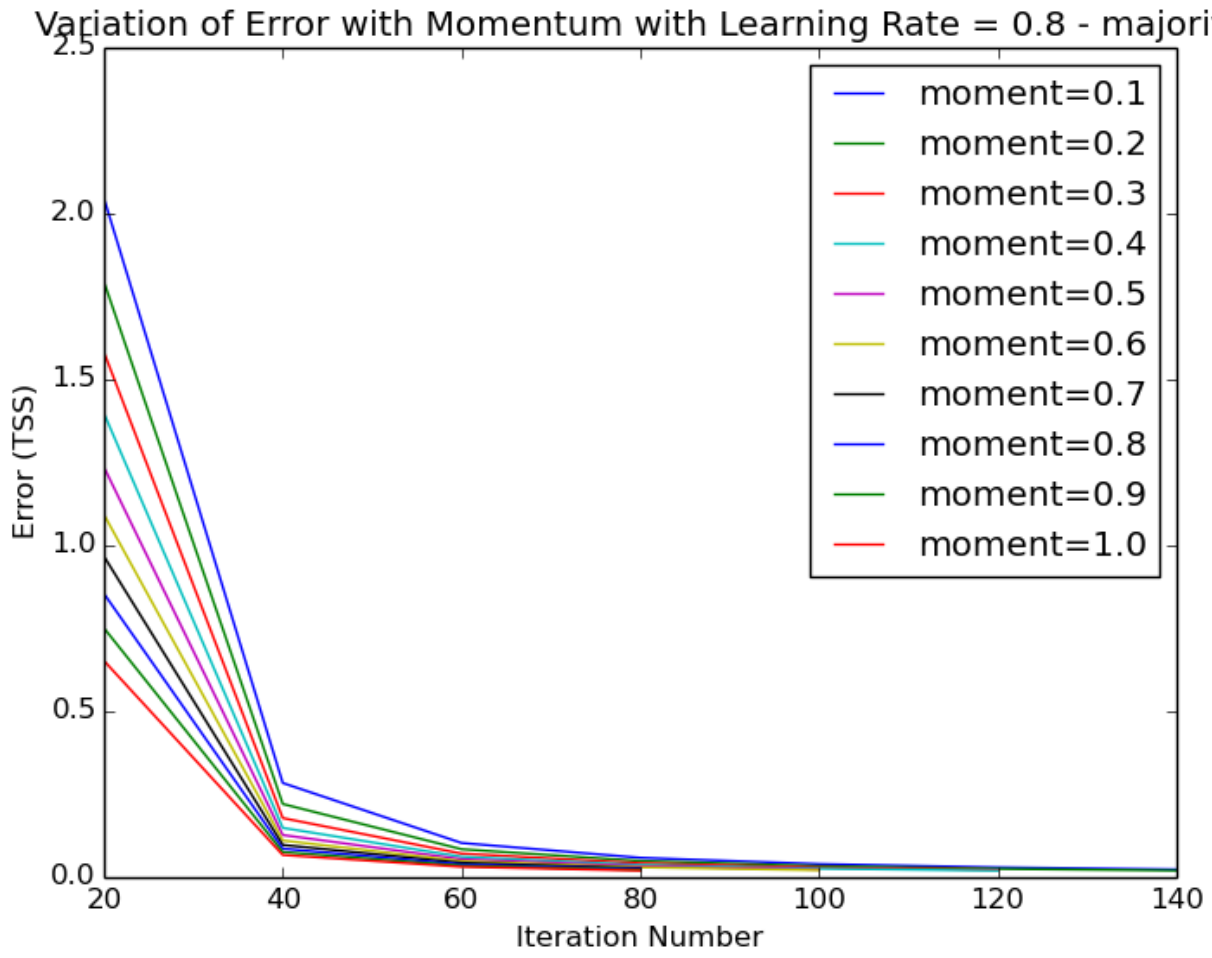
2-Input NOR


Variation of Error with Momentum with Learning Rate = 0.8 - nor
Error (TSS)
Iteration Number
0.18
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
20
40
60
80
100
120
140
160
180
moment=0.1
moment=0.2
moment=0.3
moment=0.4
moment=0.5
moment=0.6
moment=0.7
moment=0.8
moment=0.9
moment=1.0

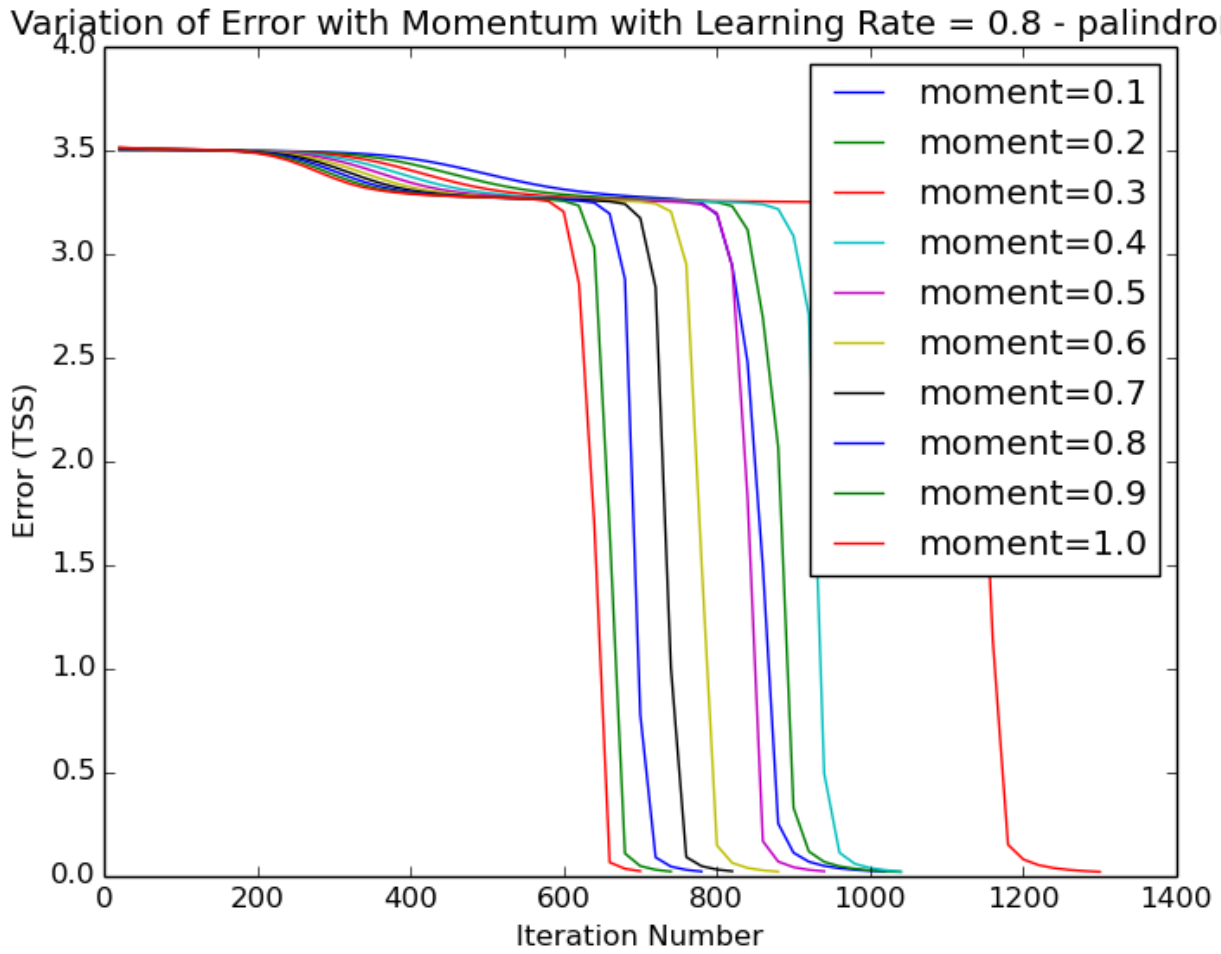
2-Input XOR


Variation of Error with Momentum with Learning Rate = 0.8 - xor
Error (TSS)
Iteration Number
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
5000
10000
15000
20000
25000
30000
moment=0.1
moment=0.2
moment=0.3
moment=0.4
moment=0.5
moment=0.6
moment=0.7
moment=0.8
moment=0.9
moment=1.0

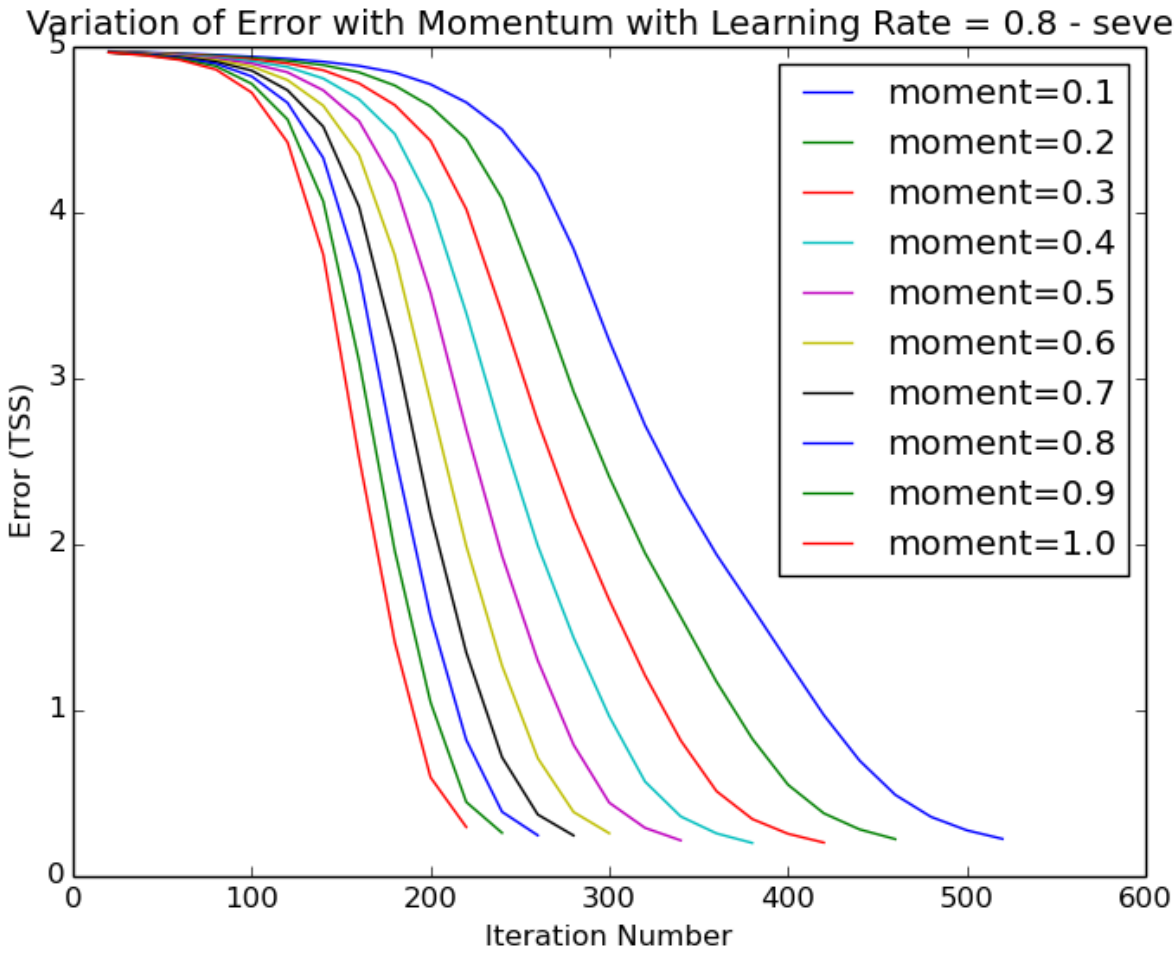
5-Input Majority



5-Input Palindrome



Seven Segment Display



Analysis of Hidden Layers

Analysis of Hidden Layers

- Number of hidden layers used :
  - 2-input XOR : 2 hidden layers
  - 5-input palindrome : 3 hidden layers

- In case of XOR, the hidden layers behaved differently for different ranges of learning rate(η).
  - For $\eta < 0.7$ and $\eta >= 0.7$, the functionality of hidden layers changes.

Analysis of Hidden Layers : XOR

- For $\eta < 0.7$

x1 \ x2	0	1
0	1	1
1	1	0

Hidden layer1 :
h1 = ¬(x1 ∧ x2)

x1 \ x2	0	1
0	0	1
1	1	1

Hidden layer2 :
h2 = x1 ∨ x2

h1 \ h2	0	1
0	X	0
1	0	1

Final layer:
O = h1 ∧ h2

Analysis of Hidden Layers : XOR

- For η >= 0.7

x1 \ x2	0	1
0	0	0
1	0	1

Hidden layer1 :
h1 = x1 ∧ x2

x1 \ x2	0	1
0	0	1
1	1	1

Hidden layer2 :
h2 = x1 ∨ x2

h1 \ h2	0	1
0	0	1
1	X	0

Final layer:
O = ¬h1 ∧ h2

Analysis of Hidden Layers : Palindrome

- For Hidden Layer 1 (h1)

x2x3 \ x4x5	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	0	0
10	1	1	0	0

x1 = 0

x2x3 \ x4x5	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	0	0
10	1	1	0	0

x1 = 1

Hidden layer1 :
h1 = x2 ∧ ¬ x4

Analysis of Hidden Layers : Palindrome

- For Hidden Layer 2 (h2)

x2x3 \ x4x5	00	01	11	10
00	0	1	1	1
01	0	1	1	1
11	0	1	1	0
10	0	1	1	0

x1 = 0

x2x3 \ x4x5	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

x1 = 1

Hidden layer2 :
h2 = x1 ∨ x5 ∨ (¬x2 ∧ x4)

Analysis of Hidden Layers : Palindrome

- For Hidden Layer 3 (h3)

x2x3 \ x4x5	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	1	0
10	0	0	1	0

x1 = 0

x2x3 \ x4x5	00	01	11	10
00	0	1	0	0
01	0	1	0	0
11	1	1	1	0
10	1	1	1	0

x1 = 1

Hidden layer3:

$h3 = \neg x4 \wedge((x2 \wedge x5) \vee (x1 \wedge x5) \vee (x1 \wedge x2)) \vee(x1 \wedge x2 \wedge x5)$

Analysis of Hidden Layers : Palindrome

- For Final Layer (O)

h1 \ h2h3	00	01	11	10
0	1	X	1	0
1	0	X	0	X

Output Layer (O):

$O = \neg x1 \wedge (\neg x2 \vee x3)$

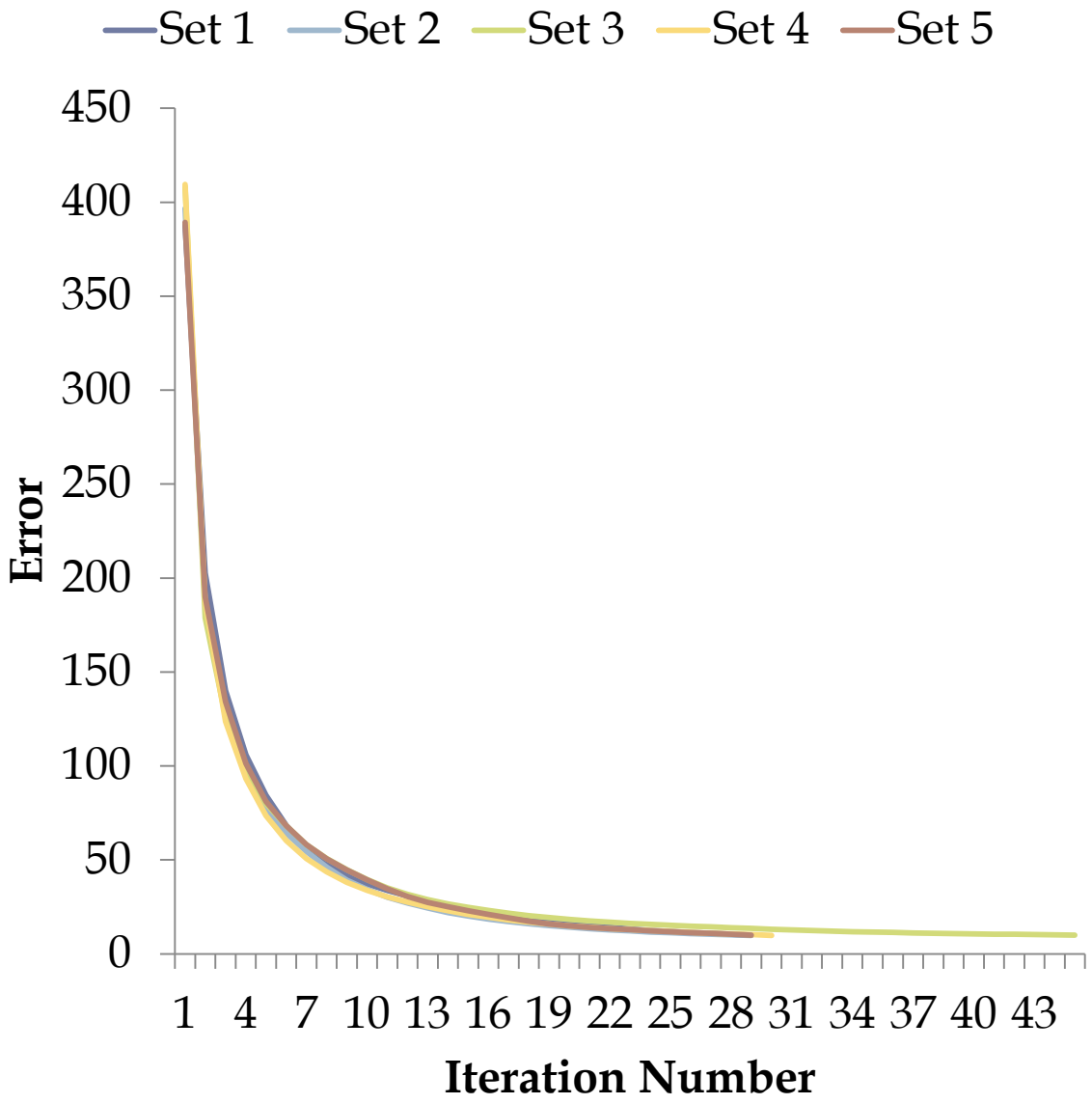
Symmetry Breaking

- If all weights are initialized to the same value then back propagation does not converge.

- Explanation – On initializing with equal weights, FFNN on each successive iteration modifies weights by equal amount. This symmetric update of weights leads to non convergence of back propagation.

Twitter Sentiment Analysis

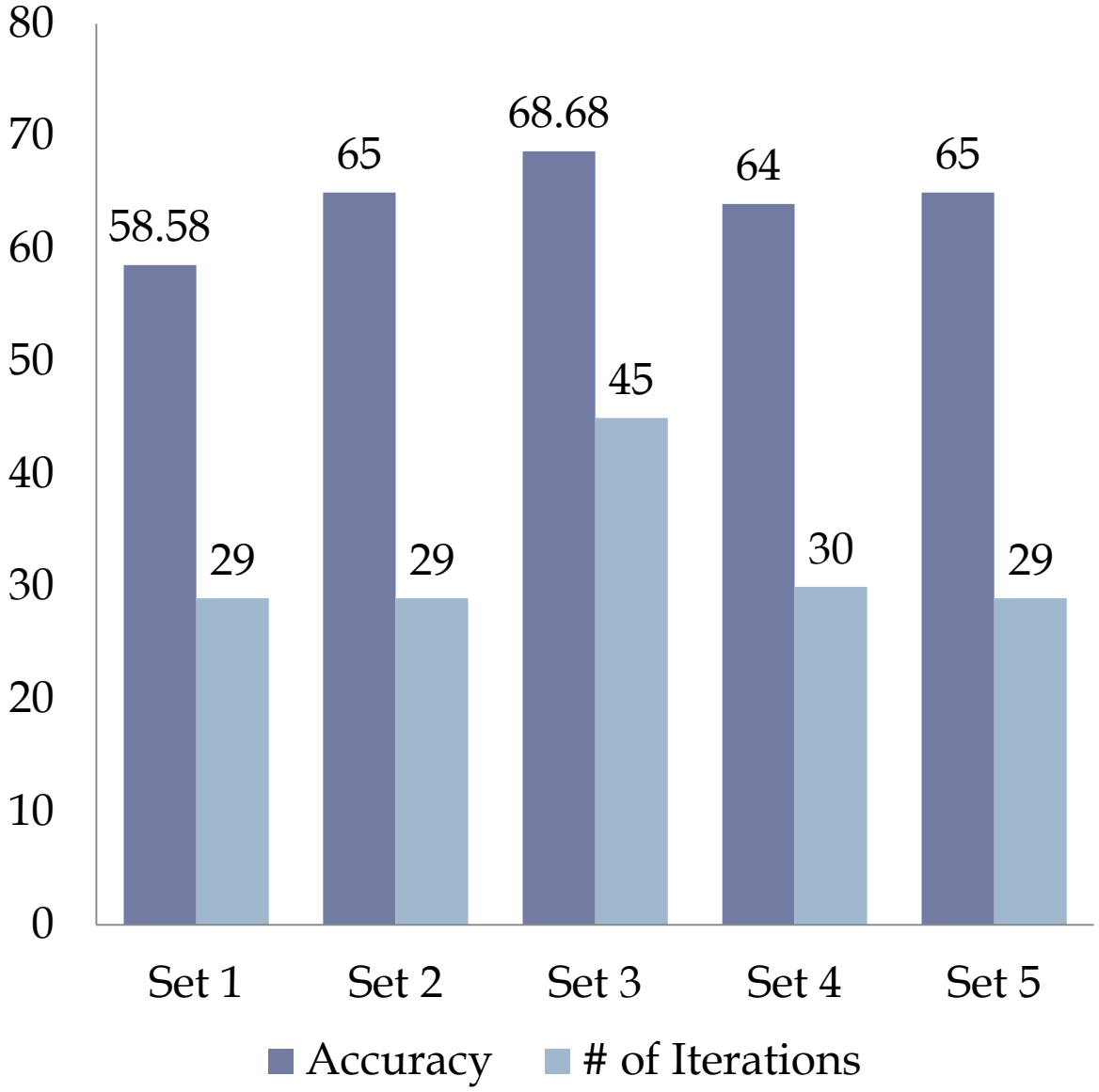
- Feature Vector Extraction
  - Separate out the words from tweets.
  - Remove duplicate and unwanted words from the list using sorting and stemming.
  - Length of feature vector = # of words extracted
  - If the word is present in the tweet, set the value 1, else 0.
- Five Fold Cross Validation
  - Shuffle the input data(tweets).
  - Separate out the data into five sets.
  - At a time, four are used for training and the remaining one is used as validation set.



Twitter Sentiment Analysis : Error v/s Iteration Number

The graph represents the variation of error with the iteration number during the training phase.

The error values are quite close for all the five folds.

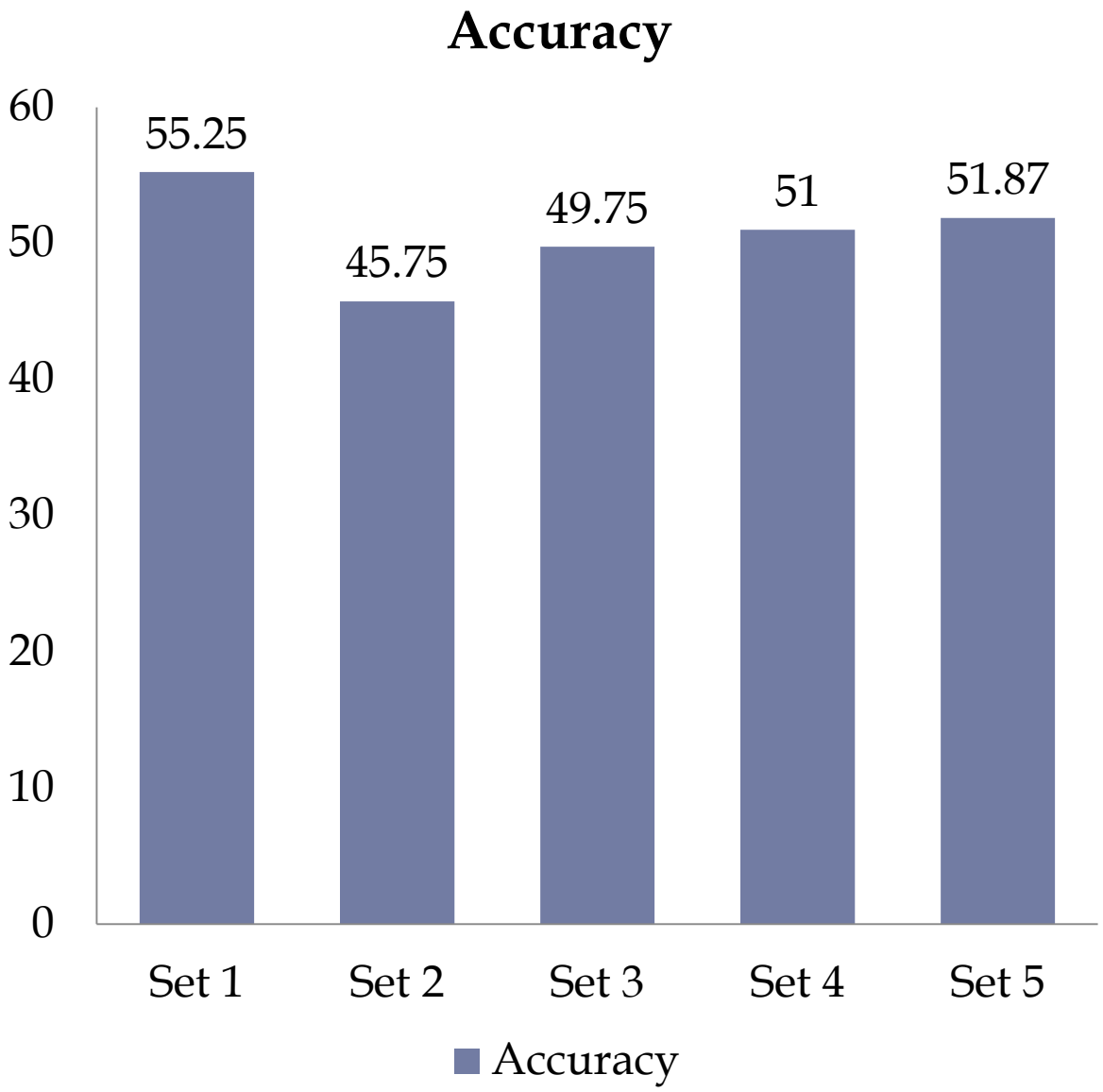


Twitter Sentiment Analysis : Accuracy and Iterations

The graph represents the accuracy on validation and # of iterations on convergence for five fold data.

Average Accuracy = 64.25%

Tweet Corpus Data



Tweet Corpus Data: Accuracy

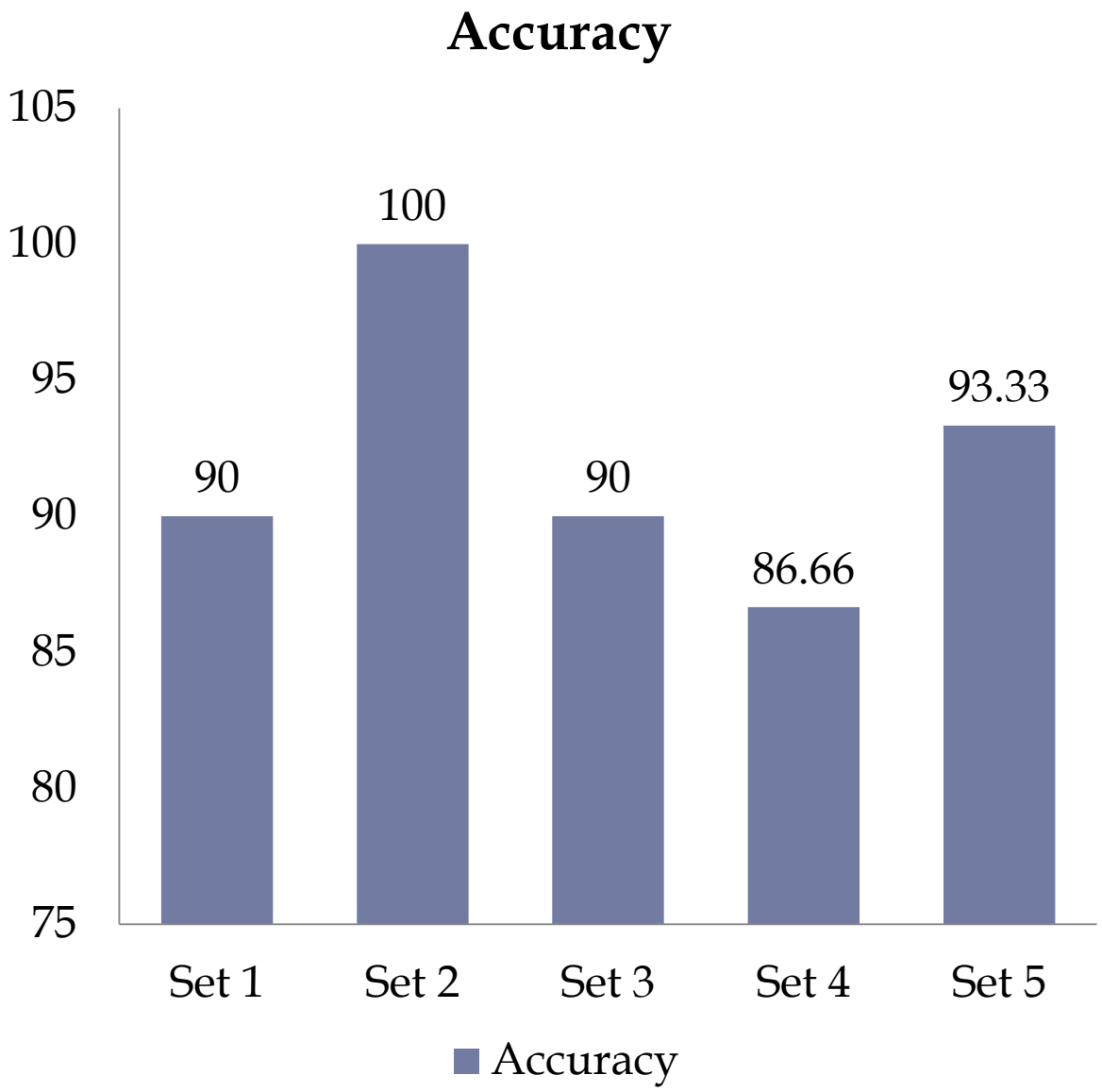
The graph represents the accuracy on validation for five fold data on new tweet corpus data.

Average Accuracy = 50.73%

Back Propagation on Benchmark Data Sets : IRIS and MONK

FFNN Configuration : IRIS Data

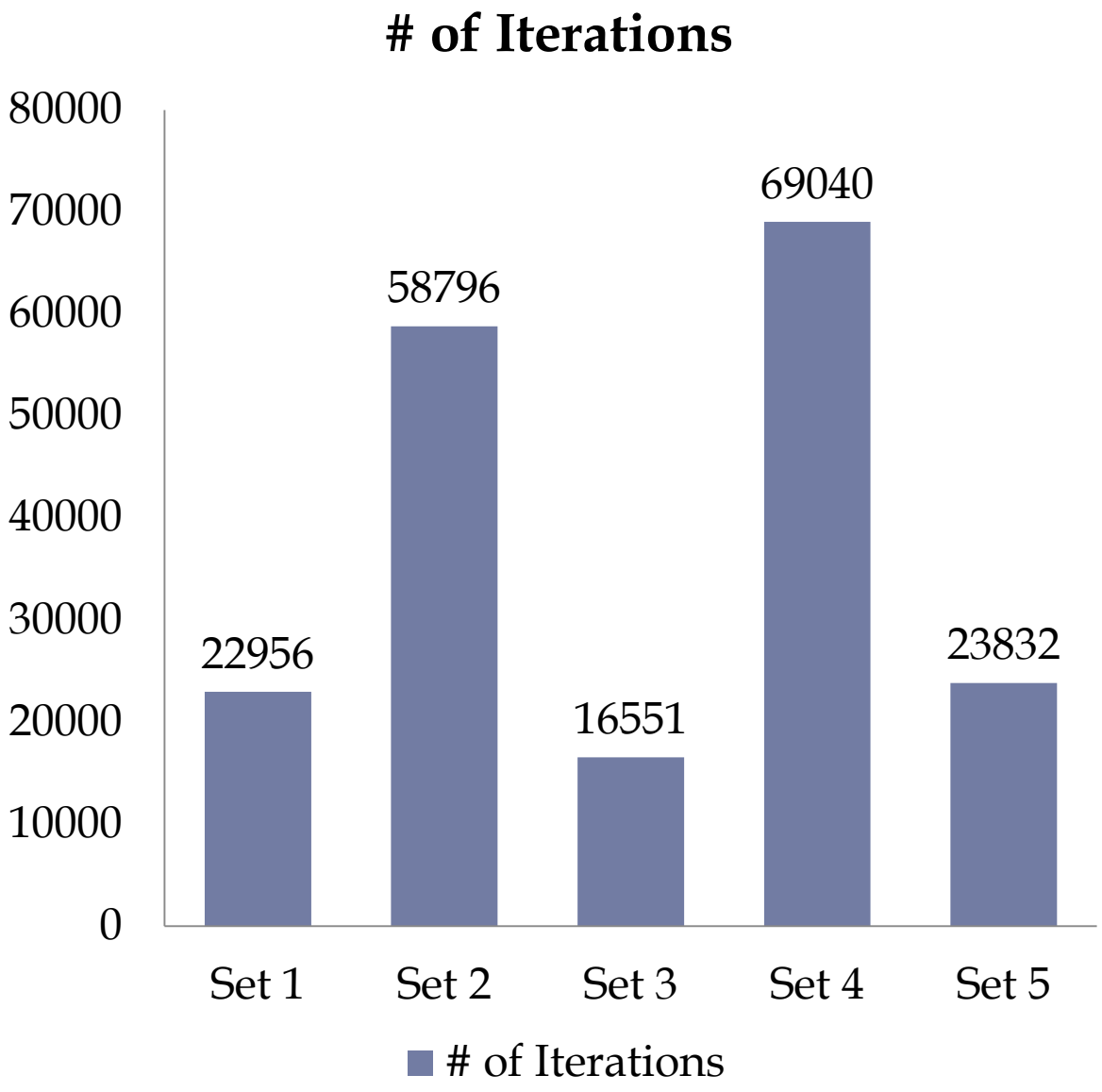
- # of inputs = 4
- # of hidden layers = 1 (with 10 neurons)
- # of outputs = 3
- Learning Rate = 0.01
- Momentum = 0.01
- Size of training data = 150



IRIS Data: Accuracy

The graph represents the accuracy on validation data sets for five fold IRIS data.

Average Accuracy = 92.00%

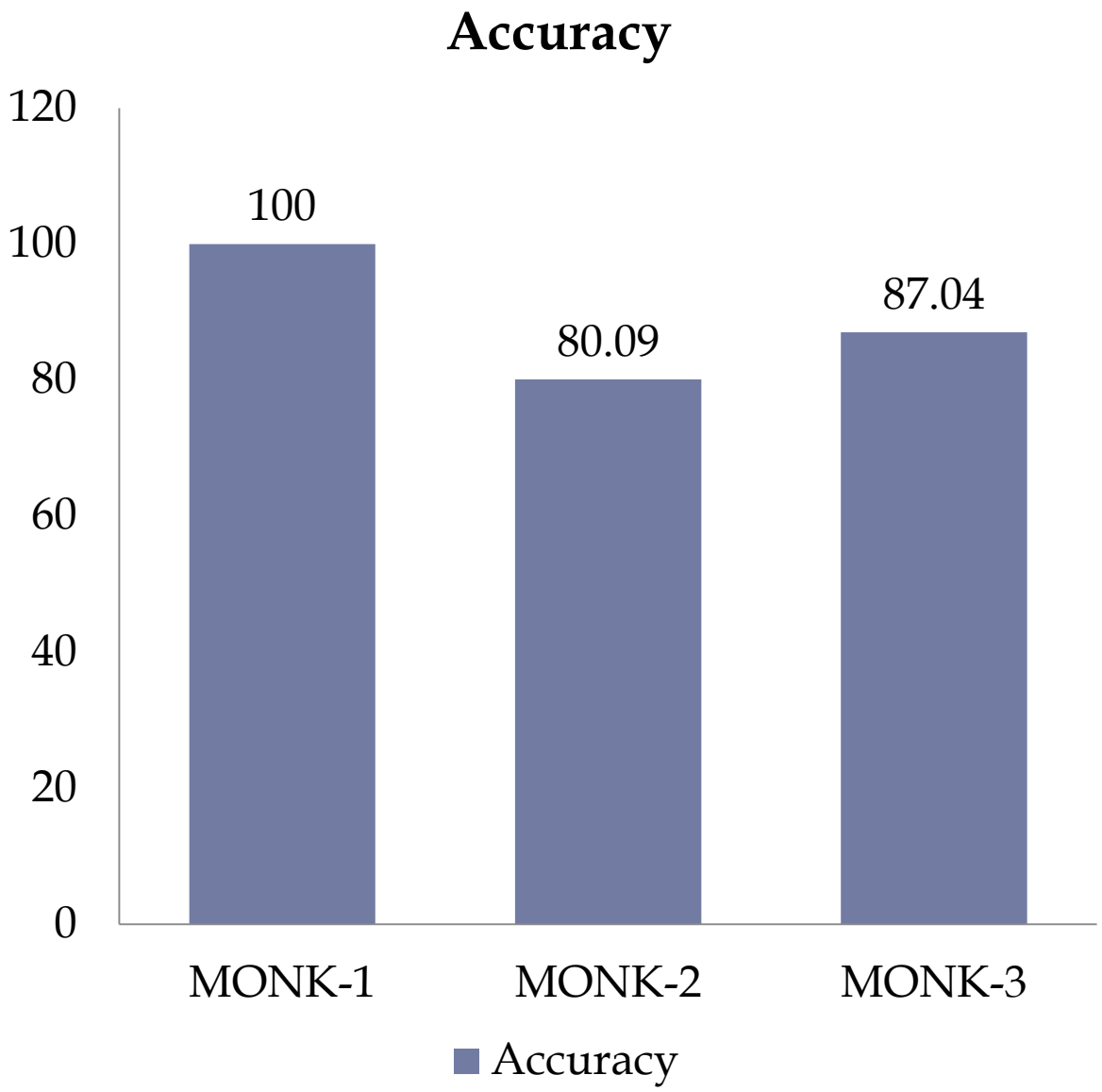


IRIS Data: # of Iterations

The graph represents the # of iterations taken for convergence of five fold IRIS data.

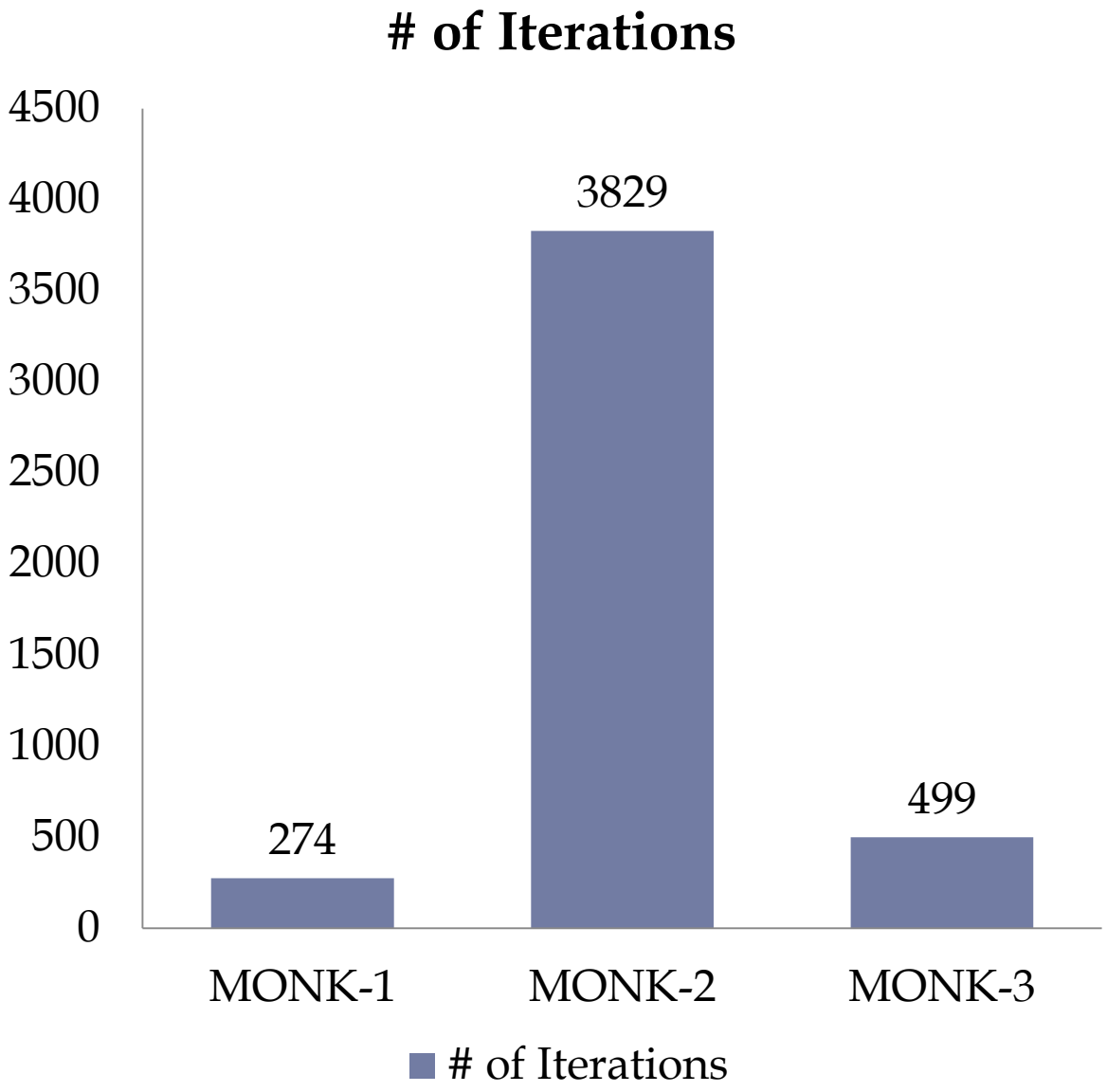
FFNN Configuration : MONK Data

- # of inputs = 6
- # of outputs = 1
- Learning Rate = 0.9
- Momentum = 0.0
- MONK-1
  - # of hidden layers = 1(with 4 neurons)
- MONK-2
  - # of hidden layers = 1(with 20 neurons)
- MONK-3
  - # of hidden layers = 1(with 7 neurons)



MONK Data: Accuracy

The graph represents the accuracy on MONK data sets (MONK-1, MONK-2, MONK-3)



MONK Data: # of Iterations

The graph represents the # of iterations taken for convergence of MONK data sets (MONK-1, MONK-2, MONK-3).

A* Algorithm

Introduction

- Implemented A* algorithm and applied it on the following problems :
  - 8-tiles Puzzle Problem
  - Missionaries and Cannibal Problem
- Implemented bidirectional search and compared it with the A* algorithm.
- Compared performance of various heuristics with both A* and Bidirectional search.

A* Implementation

- Created a class called “Node” consisting of following data members :
  - ID: unique identifier for node, represents state.
  - isVisited: set when a node is removed from open list
  - parent : pointer to the parent node
  - gvalue and hvalue
  - Compute_hvalue( ): Returns hvalue according to the heuristic
  - Get_children( ): Returns all possible states obtained from the node after applying one operation on that node

A* Implementation

- Parent Pointer Redirection
  - Whenever an alternate shorter path to an already visited node is found, the *parent pointer* of that node and its *gvalue* is readjusted.
  - The *gvalue* of the descendants of that node is also updated.
  - Parent pointer redirection ensures that already visited node i.e. a node is in closed list is never put again into the open list.

8-Tiles Puzzle Problem

- Configuration
  - The eight puzzle consists of a three by three board with eight numbered tiles and a blank space.
- Operation
  - A tile adjacent to the blank space can slide into the space.
- Objective
  - The object is to figure out the steps needed to get from one configuration of the tiles to another.

8-Tiles Puzzle Problem : Heuristics

- H0 : Trivial Heuristic
  - *hvalue = 0*
  - Satisfies admissibility of A* .
  - Follows Montone Restriction.

- H1 : No. of Displaced Tiles
  - *hvalue* = No. of tiles which are displaced from their actual position in the goal node.
  - Satisfies admissibility of A* .
  - Follows Monotone Restriction.

.

8-Tiles Puzzle Problem : Heuristics

- H2 : Manhattan Distance
  - *hvalue* = sum of manhattan distances of tiles from their destined position.
  - Satisfies admissibility of A* .
  - Follows Monotone Restriction.

- H3 : No. of inversion pairs/2
  - *hvalue* = abs([#Inv_pairs(Node) - #Inv_pairs(Goal)]/2)
  - Satisfies admissibility of A* .
  - Follows Monotone Restriction.

H3 : Proof of admissibility and MR

- One operation i.e. moving a tile to the blank space results in change in number of inversion pairs either by 0 or 2.

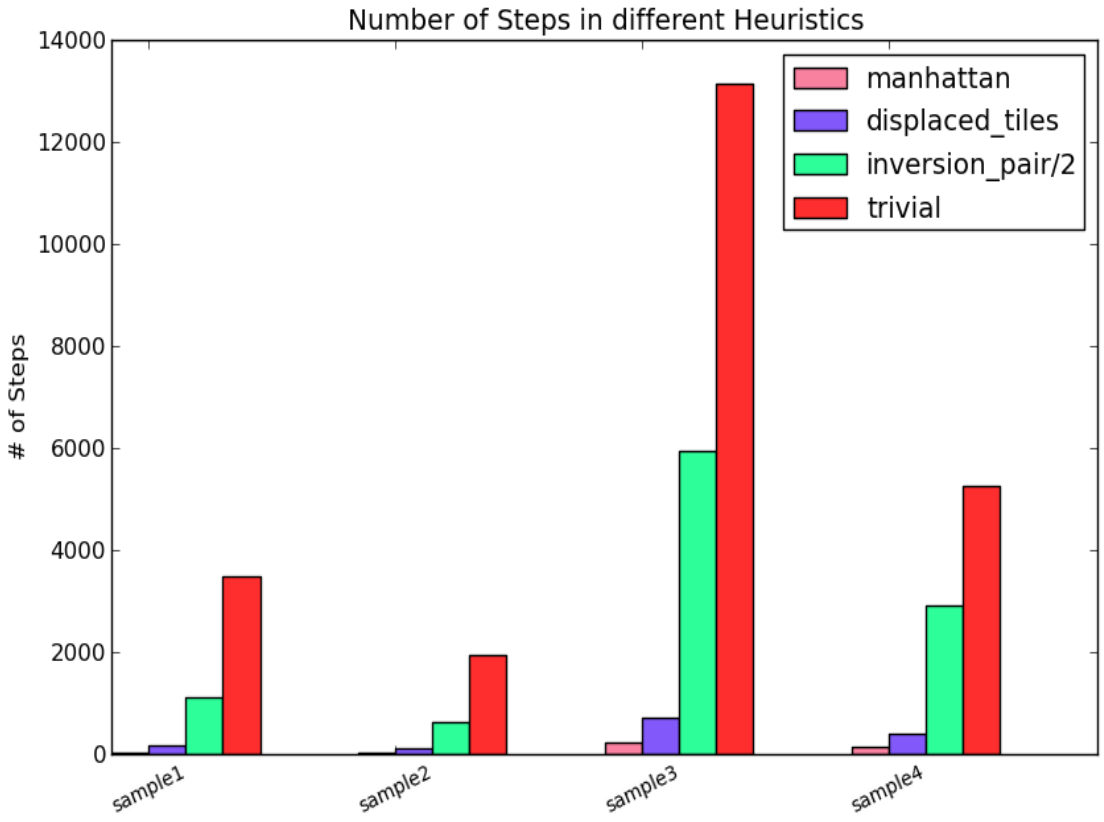
- For MR satisfiability,
  - $h(p) <= h(c) + cost(p,c)$
  - Here, h(n) = #inversion_pairs(n)/2
  - Cost(p, c) = 1

- Also, MR => Admissibility

Proof of Monotone Restriction

- Case 1:
  - #inversion_pairs(c) = #inversion_pairs(p) + 2
  - h(c) = h(p) + 1
  - Then, h(p) <= h(c) + cost(p,c) is satisfied trivially.
- Case 2:
  - #inversion_pairs(c) = #inversion_pairs(p) - 2
  - h(c) = h(p) - 1
  - Then, h(p) = h(c) + cost(p,c)
  - Thus, h(p) <= h(c) + cost(p,c) is satisfied.
- Case 3:
  - #inversion_pairs(c) = #inversion_pairs(p)
  - h(c) = h(p)
  - Then, h(p) <= h(c) + 1 is satisfied.
- H3 satisfies MR.

- Hence, Heuristic H3 is admissible.



8-Tiles : Better Heuristic Performs Better

1. Comparison of various heuristics employed for 8-tiles problem.

2. Manhattan distance performs best whereas the trivial heuristic is the poorest.

3. The inversion_pairs/2 heuristic performs worse than Manhattan and displaced tiles.

8-Tiles Puzzle Problem : Heuristics

- H4 : No. of inversion pairs
  - *hvalue* = abs[#Inv_pairs(Node) - #Inv_pairs(Goal)]
  - Violates admissibility of A*.
  - Violates Monotone Restriction.

1	2	3
4	5	
7	8	6

1	2	3
4	5	6
7	8	

  - h(n) = 2, h*(n) = 1 h(goal) = 0
  - Since h(n) > h*(n), H4 is inadmissible.

8-Tiles: Parent Pointer Redirection with H4

- Keeping the final state = [[1,2,3],[8,0,4],[7,6,5]]

Start State	Parent Pointer Direction Happens
[[3,5,8],[4,1,6],[2,7,0]]	✓
[[3,1,4],[6,2,8],[0,5,7]]	✓
[[8,3,5],[4,1,6],[2,7,0]]	✓
[[2,8,1],[4,6,3],[0,7,5]]	×
[[1,2,3],[6,5,4],[0,7,8]]	×
[[1,3,2],[4,0,8],[7,6,5]]	×

8-Tiles Puzzle Problem : Heuristics

- H5 : modified displaced tiles
  - *hvalue* =[#displaced tiles w.r.t. start state]*20 + 20
  - Violates admissibility of A*.
  - Violates Monotone Restriction.

1	2	3
4		5
7	8	6

start state

1	2	3
4	5	
7	8	6

n

1	2	3
4	5	6
7	8	

final state

- h(n) = 1*20+20 = 40 , h*(n) = 1
- Since h(n) > h*(n), H5 is inadmissible.

Non-optimality of H5

- Start state = [[0, 1, 2][3, 4, 5][6, 7, 8]]
- Final state = [[4, 1, 2][0, 6, 3][7, 5, 8]]
- With manhattan distance heuristic
  - #steps = 114
  - Pathlength = 16
- With H5 heuristic
  - #steps = 1085
  - Pathlength = 18
- H5 gives sub-optimal path.

Non-optimality of H5

- Start state = [[1, 3, 6], [4, 0, 2], [7, 5, 8]]
- Final state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]
- With trivial heuristic i.e. h = 0
  - #steps = 108
  - Pathlength = 7
- With H5 heuristic
  - #steps = 101
  - Pathlength = 7
- H5 gives optimal path in smaller number of steps.

8-Tiles Puzzle : Non-Reachability

- One operation i.e. moving a tile to the blank space results in change in number of inversion pairs either by 0 or 2.
- Therefore, the difference between the number of inversion pairs of the start configuration and the goal configuration should be even for reachability.
- If the difference between the number of inversion pairs is odd, then the 8-puzzle problem is not solvable.

Non-Reachability : Example

8	3	5
4	6	1
2	7	

Start state

1	2	3
8		4
7	6	5

Final State

- No. of inversion pairs in start state = 16
- No. of inversion pairs in final state = 7
- ➔ Difference = 9, which is odd.
- Hence, Not solvable.

Missionaries and Cannibal Problem

- Configuration
  - <#M, #C, P>
  - #*M* = Number of missionaries on bank *L*
  - #*C* = Number of cannibals on bank *L*
  - *P* = Position of the boat
- Constraint
  - For both banks, if there are missionaries present on the bank, they cannot be outnumbered by cannibals (if they were, the cannibals would eat the missionaries).
- Objective
  - How can the boat be used to safely carry all the missionaries and cannibals across the river?

Missionaries and Cannibals: Heuristics

- H0 : Trivial Heuristic
  - *hvalue = 0*
  - Satisfies admissiblity of A* .
  - Follows Montone Restriction.

- H1 : No. of persons left on L bank / 2
  - *hvalue = (#M + #C)/2*
  - Satisfies admissiblity of A* .
  - Follows Montone Restriction.

H1 : Proof of admissibility

- A boat can carry at most two persons at a time.
- The number of persons at the left bank = #M + #C.
- Therefore,
  - Min # steps required = (#M + #C)/2
  - Also, h*(n) >= min steps required.
  - h(n) = min steps required = (#M + #C)/2.
  - Therefore, h(n) <= h*(n).

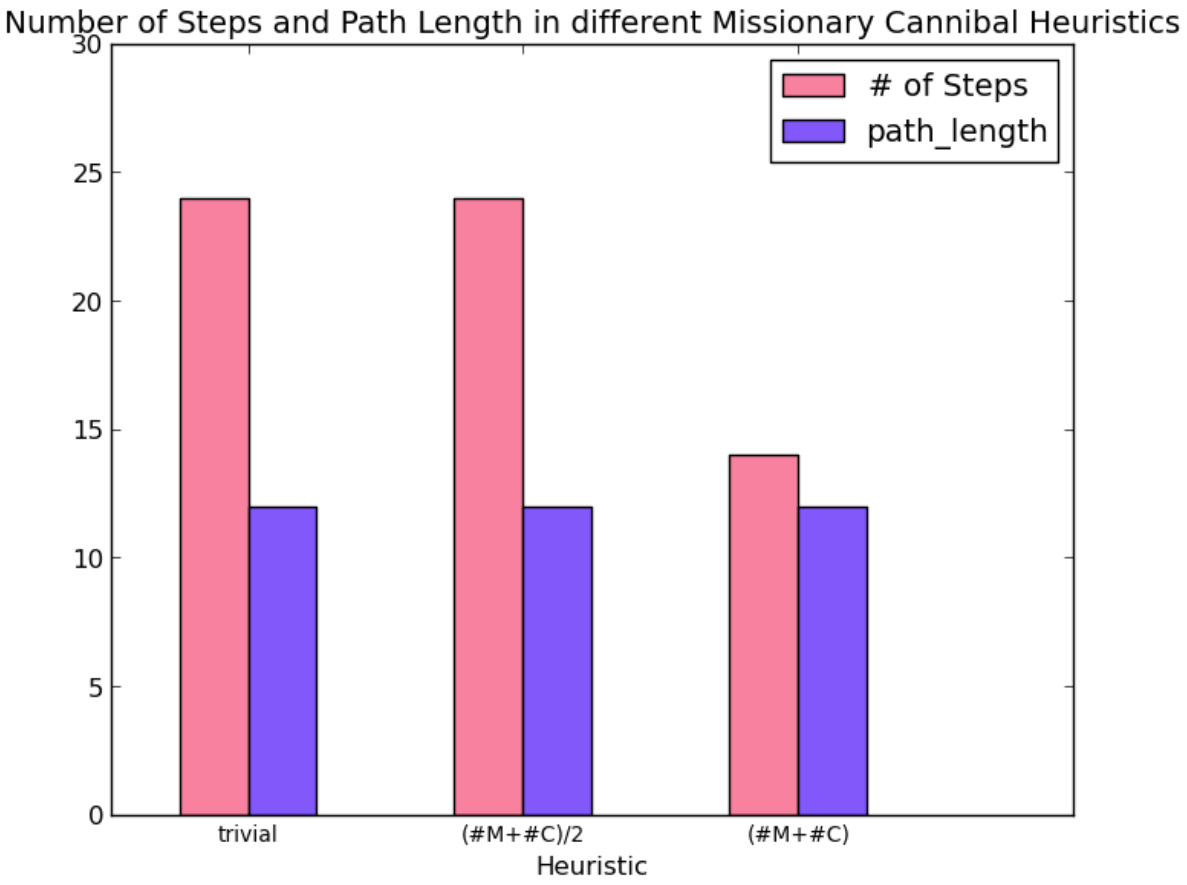
- Hence, Heuristic H1 is admissible.

Missionaries and Cannibals: Heuristics

- H2 : No. of persons left on L bank
  - *hvalue* = *(#M + #C)*
  - Violates admissiblity of A* .
  - Violates Montone Restriction.

- Let the configuration of node n be <1, 1, L >
  - Then, h(n) = 2
  - Here, h*(n)=1 since M and C can be carried in a single boat safely.

- Hence, h(n) > h*(n) violates admissibility of H2.



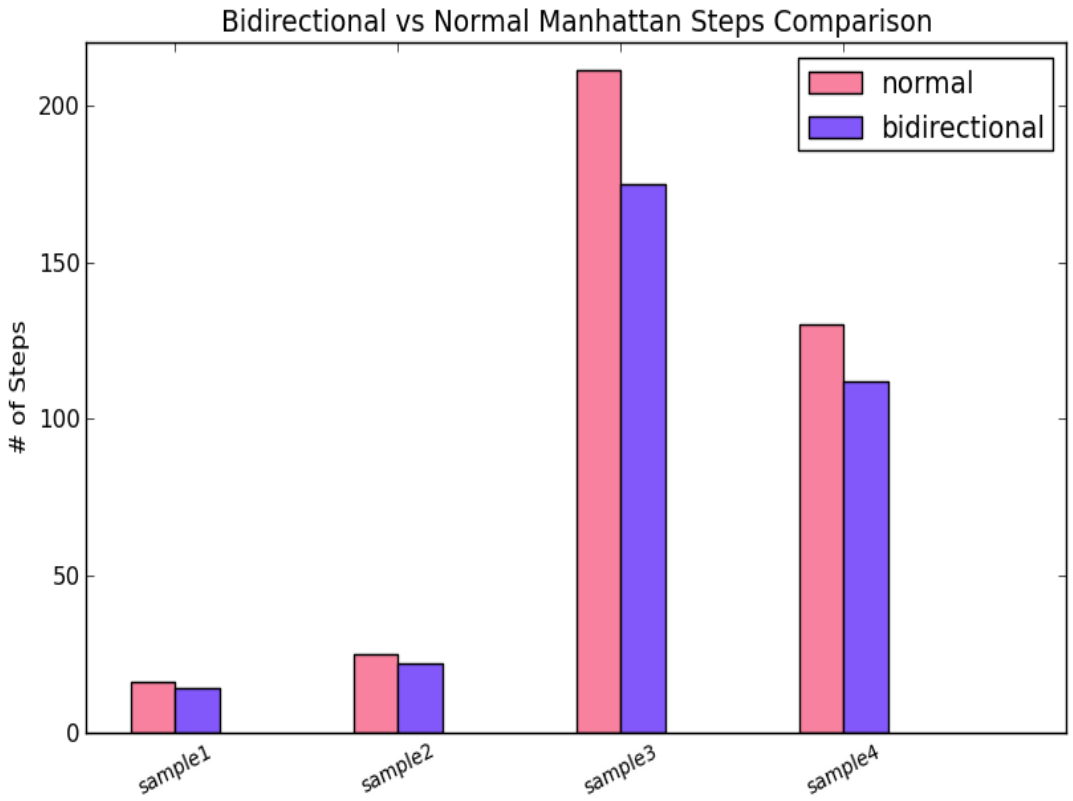
Missionary-Cannibal Heuristics

1. The trivial heuristic(H0) as well as '#persons/2' (H1) take the same no. of steps and same path length.

2. '#persons' (H2), though inadmissible, gives the optimal path as well as takes fewer steps to converge.

Bidirectional Search

- Runs two simultaneous searches :
  - One forward from initial state
  - One backward from the goal state
- Termination Condition
  - The search is terminated when the two search frontiers meet each other.
  - Construct the single path extending from start node through intersection node to the goal.
- Bidirectional search does not guarantee optimality.

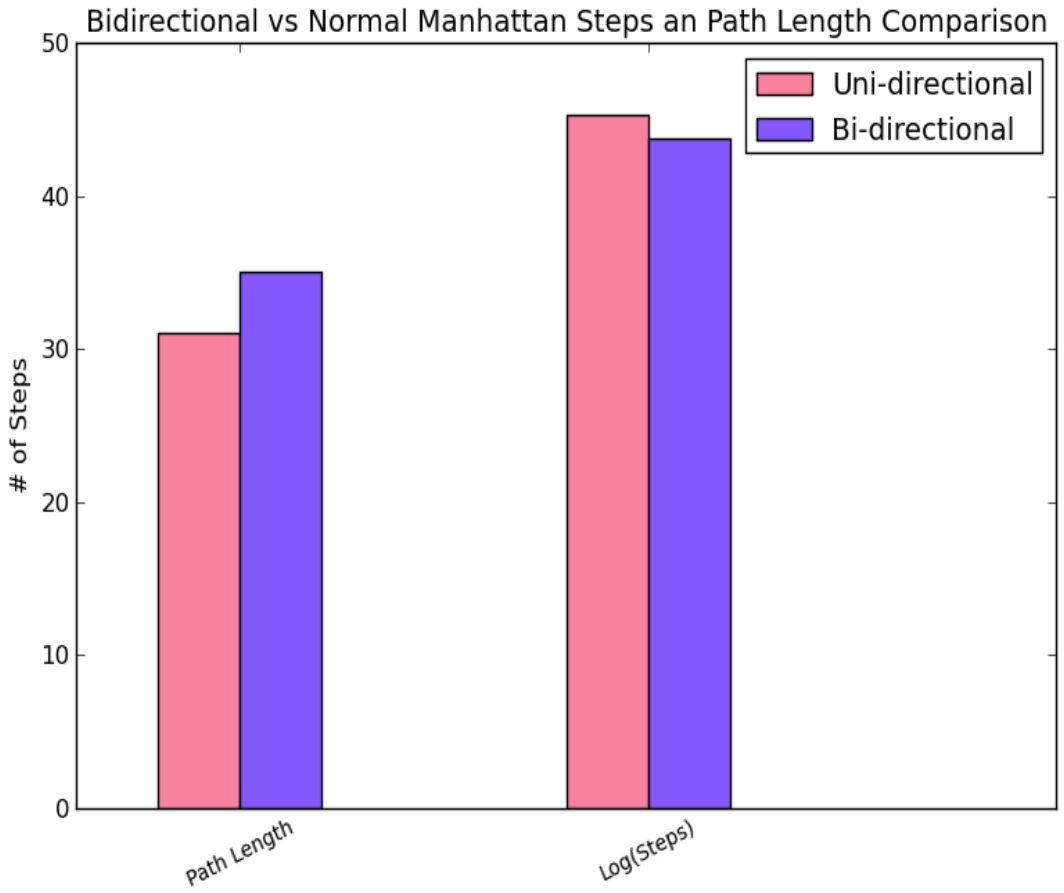


Bidirectional Search : 8-Tiles

1.Bidirectional search converges faster than even the Manhattan heuristic.

2. It follows from previous results that bidirectional search will converge faster than other heuristics as well.

3. However, the optimality is not guaranteed.



Non-optimality of Bidirectional Search : 8-Tiles

1. Start state = [[5,6,7],[4,0,8],[3,2,1]]
Final state = [[1,2,3],[8,0,4],[7,6,5]]

2. Unidirectional search : Steps = 3837; PL= 31

Bidirectional search : steps = 2925; PL= 35

3. Bidirectional Search converges faster than the Manhattan heuristic but violates optimality.

Theorem Prover

Introduction

- Implemented an Automatic Theorem Prover based on Hilbert's Formalization.
- Implemented Heuristics to apply one of the three axioms of formal system whenever needed.
- Implemented a feature for human help in case proof getting stuck at some step (when algorithm can't decide which axiom would work).

Deduction Theorem

- If A1, A2, A3..... An $\vdash$ B

  then

  A1, A2, A3, ...An-1 $\vdash$ An $\rightarrow$ B

- In our theorem we have used Deduction theorem to generate initial set of hypothesis.

- Once hypothesis set is generated then we work solely with the First Principle approach and try to derive F.

Theorem Prover : Implementation

- Input to the Theorem Prover is a well formed formula which needs to be proved.
- The given input formula is parsed as follows
  - A well formed formula (WFF) is either a literal, special symbol (F) i.e. WFF is atomic or
  - WFF → WFF
- We represent a WFF using a class structure which has id (string representation of formula), left component and right component.

Implementation continued..

- We maintain a formula list which consists of
  - Hypothesis generated using Deduction theorem on input formula
  - Results of Modus Ponens
  - Results of Axioms

- Initially formula list consists of only Hypothesis set which is later augmented by applying MP rules and Axioms on entities already present in formula list.

Implementation : Algorithm

- Given : Formula List (FL) containing Hypothesis Set
- Goal : To derive **F**
- Procedure :

begin

if **F** is not present in Formula List

apply Modus Ponens.

if MP succeeds

add new WFF to FL and **repeat**

apply heuristics for axiom generation

if heuristic succeeds

add new WFF to FL and **repeat**

ask for human help

repeat

end

Implementation : Modus Ponens

- We iterate in the formula list looking for a WFF which matches with the left component of another WFF in formula list.
- If so happens, we apply MP rule on those two WFFs and generate a new WFF which is then added to the formula list.
- If no such WFFs are found then we either use heuristics to apply Axioms or take human help.

Heuristic : Axiom-1

- We apply Axiom 1 on a WFF (w1) if there exists another WFF (w2) such that the right component of its left component is w1.

- So we generate a WFF as (w1 → (w2.left.left → w1)) This enables us to derive w2.right using MP

- For example, w1 : A, w2 : ((B → A) → C)
  - On applying Axiom 1 we get w3 : (A → (B → A))
  - Now using MP on w2 and w3 we derive w4 : C

Heuristic : Axiom-2 and Axiom-3

- If any of the WFF (w1) in formula list matches the pattern of left component of axiom-2 or axiom-3 then we apply that axiom to generate a new WFF with w1 as its left component.

- For example, w1 : (P → (Q → R))
  - Axiom-2 : (A → (B → C)) → ((A → B) → (A → C))
  - New WFF : (P → (Q → R)) → ((P → Q) → (P → R))

Heuristic: Exhaustive Axiom Generation

- We define 'Level' of WFF as the number of '→' occurring in that WFF.
- For example,
  - WFF_{L0} : A, B
  - WFF_{L1} : (A→B), (C→F)
- We pick up all the WFFs of level-0 and level-1 from the formula list and replace the literals occurring in an axiom by such WFFs.
- This results in the generation of a set of new WFFs. If any of new WFF is not already present in the FL then it is added to FL.

Exhaustive Axiom Generation: Example

- Formula List : [P,Q,(P→Q)]
- Applying Axiom-1 i.e. (A→(B→A)), we get:
  - (P→(Q→P))
  - (Q→(P→Q))
  - (P→((P→Q)→P))
  - (Q→((P→Q)→Q))
  - ((P→Q)→(P→(P→Q)))
  - ((P→Q)→(Q→(P→Q)))

Human Help

- When MP rule and every heuristic fails to generate any new WFF, human help is sought out.

- User is asked to apply one of the axioms by entering the axiom number and fixing the value of each literal occurring in that axiom.

- For example,
  - Axiom # : 2; i.e. (A → (B → C)) → ((A → B) → (A → C))
  - Enter value of A : (P→Q)
  - Enter value of B : Q
  - Enter value of C : (P→F)

Prolog
Circuit Simulator

Introduction

- Implemented a circuit simulator in Prolog and verified using following circuits
  - 3 input Full Adder
  - 5 input Palindrome
  - 5 input Majority

- Implemented Multi-Input AND, OR, XOR gates and single input NOT gate.

Input Specification

- Description of Input wires:
  - value(wire_id, value).
- Description of a gate (circuit element):
  - ckt_elem(elem_id).
  - type(elem_id, gate_type).
  - inputs(elem_id, number_of_inputs).
- Description of connections:
  - connected(wire_id1, wire_id2).
  - connected(in(input_num, elem_id), wire_id).
  - connected(out(elem_id), wire_id).

Implementation : Predicates Used

- are_connected(T1, T2)
  - To check if wire T1 and wire T2 are connected.
- wire(T)
  - To check if T is a wire
  - wire can be an input value, an output port or input port of a circuit element
- elem_def(G, N)
  - To get number of inputs of circuit element G in N
- all(G, X)
  - To check if all inputs of circuit element G has value X
- count(G)
  - To check if number of inputs with value 1 are odd

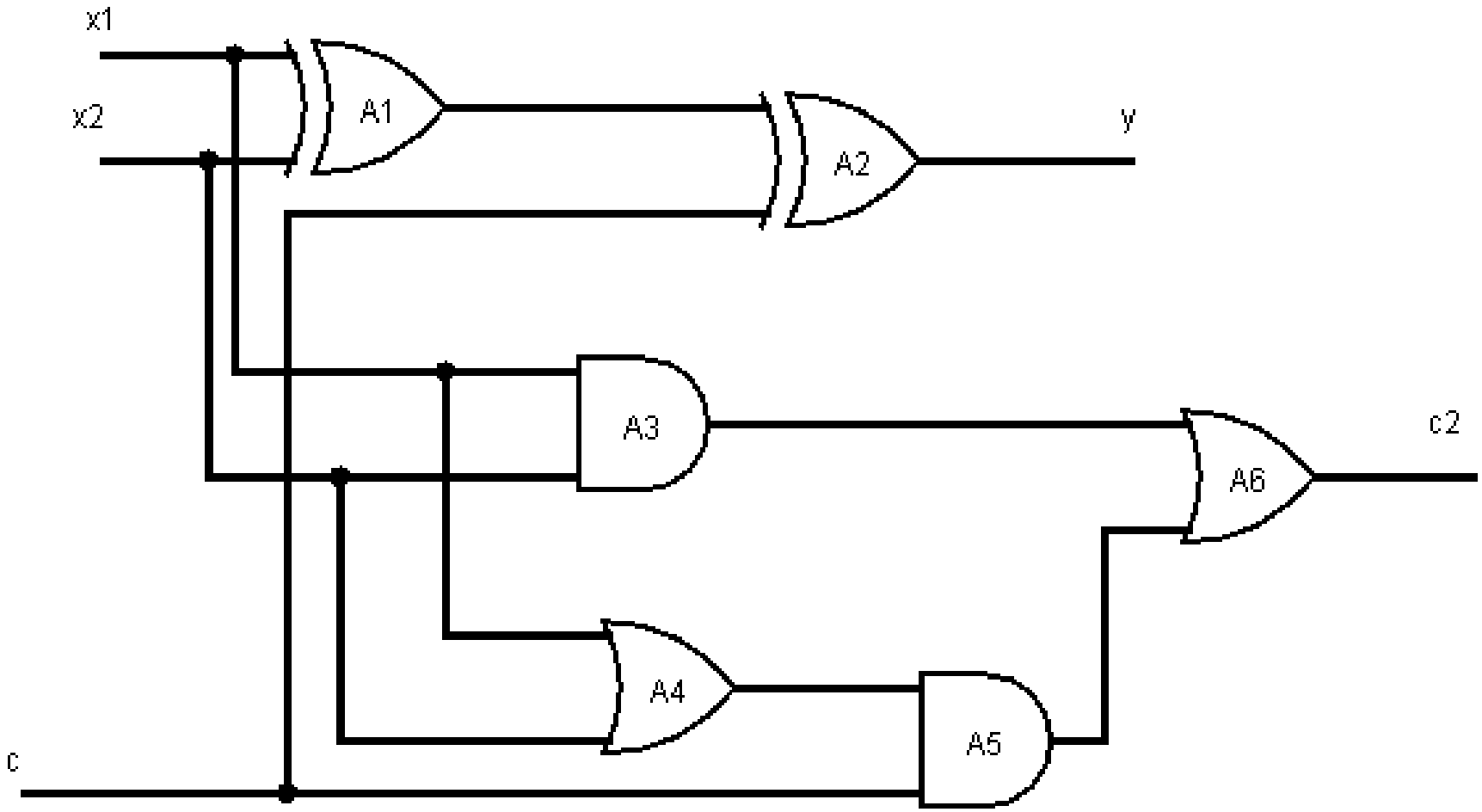
Implementation : Predicates Used

- signal(T, X)
  - If T is an assigned value then X gets its value
  - If T is an output of a circuit element i.e. out(G) then it is evaluated accordingly
    - AND Gate –
      - 1 if all(G, 1), 0 otherwise
    - OR Gate –
      - 1 if all(G, 0) , 0 otherwise
    - NOT Gate –
      - 1 if input is 0, 0 otherwise
    - XOR Gate –
      - 1 if count(G) , 0 otherwise
  - If T is a wire, and there is another wire connected to it then assign the same value to it

Full Adder Truth Table

X1	X2	C1	Y	C2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Full Adder Circuit



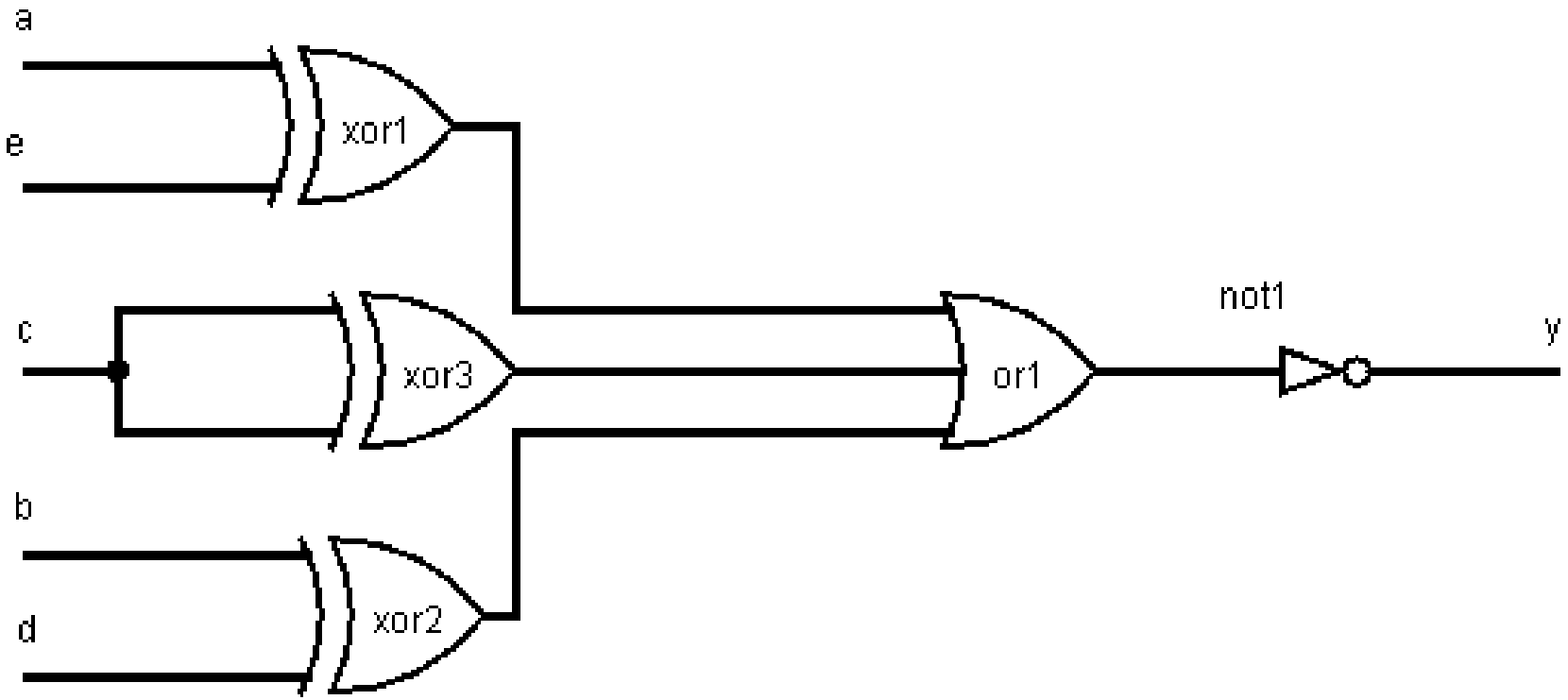
Palindrome Truth Table

A	B	C	D	E	Y
0	0	0	0	0	**1**
0	0	0	0	1	**0**
0	0	0	1	0	**0**
0	0	0	1	1	**0**
0	0	1	0	0	**1**
0	0	1	0	1	**0**
0	0	1	1	0	**0**
0	0	1	1	1	**0**
0	1	0	0	0	**0**
0	1	0	0	1	**0**
0	1	0	1	0	**1**
0	1	0	1	1	**0**
0	1	1	0	0	**0**
0	1	1	0	1	**0**
0	1	1	1	0	**1**
0	1	1	1	1	**0**

Palindrome Truth Table

A	B	C	D	E	Y
1	0	0	0	0	**0**
1	0	0	0	1	**1**
1	0	0	1	0	**0**
1	0	0	1	1	**0**
1	0	1	0	0	**0**
1	0	1	0	1	**1**
1	0	1	1	0	**0**
1	0	1	1	1	**0**
1	1	0	0	0	**0**
1	1	0	0	1	**0**
1	1	0	1	0	**0**
1	1	0	1	1	**1**
1	1	1	0	0	**0**
1	1	1	0	1	**0**
1	1	1	1	0	**0**
1	1	1	1	1	**1**

Palindrome Circuit



Majority Truth Table

A	B	C	D	E	Y
0	0	0	0	0	**0**
0	0	0	0	1	**0**
0	0	0	1	0	**0**
0	0	0	1	1	**0**
0	0	1	0	0	**0**
0	0	1	0	1	**0**
0	0	1	1	0	**0**
0	0	1	1	1	**1**
0	1	0	0	0	**0**
0	1	0	0	1	**0**
0	1	0	1	0	**0**
0	1	0	1	1	**1**
0	1	1	0	0	**0**
0	1	1	0	1	**1**
0	1	1	1	0	**1**
0	1	1	1	1	**1**

Majority Truth Table

A	B	C	D	E	Y
1	0	0	0	0	**0**
1	0	0	0	1	**0**
1	0	0	1	0	**0**
1	0	0	1	1	**1**
1	0	1	0	0	**0**
1	0	1	0	1	**1**
1	0	1	1	0	**1**
1	0	1	1	1	**1**
1	1	0	0	0	**0**
1	1	0	0	1	**1**
1	1	0	1	0	**1**
1	1	0	1	1	**1**
1	1	1	0	0	**1**
1	1	1	0	1	**1**
1	1	1	1	0	**1**
1	1	1	1	1	**1**

Majority Circuit